Modeling Cloth from Examples

Ryan White

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-134
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-134.html

November 8, 2007

Acknowledgement

This research was performed under an appointment to the Department of Homeland Security (DHS) Scholarship and Fellowship Program, administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and DHS. ORISE is managed by Oak Ridge Associated Universities (ORAU) under DOE contract number DE-AC05-06OR23100. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORAU/ORISE.

This research was also partially supported by the ATI Graduate Research Fellowship.

Modeling Cloth from Examples

by

Ryan M White

B.S. (California Institute of Technology) 2002
M.S. (University of California, Berkeley) 2005

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering-Electrical Engineering and Computer Science

and the Designated Emphasis

in

Communication, Computation, and Statistics

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David Forsyth, Co-Chair
Professor Jitendra Malik, Co-Chair
Professor Panayiotis Papadopoulos
Professor James O'Brien

Fall 2007

Modeling Cloth from Examples

Abstract

Modeling Cloth from Examples

by

Ryan M White

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Science

University of California, Berkeley

Professor David Forsyth, Co-Chair

Professor Jitendra Malik, Co-Chair

We measure cloth properties and create cloth animations from video of actual cloth. This work spans several domains: texture tracking and replacement, 3D shape estimation from a single view, 3D shape from multiple views and data driven cloth animation.

In the first part of the thesis, the emphasis is on single view estimation of visual properties and the related problem of texture replacement. We show that *screen print* cloth has advantageous properties and that simple lighting estimation makes texture replacement and geometry estimation substantially better.

In the second part of the thesis, we create cloth animations in a data-driven manner by recording real cloth movement. Our method involves printing custom clothing, capturing video of the clothing from multiple viewpoints and then building models of the cloth motion as frame-by-frame geometry. This problem is difficult because of occlusion:

often some portion of the cloth makes it impossible to see other regions of the cloth. To overcome this challenge, we print a multi-colored pattern on the surface of the cloth to simplify identification of the surface and we use a data-driven hole filling technique to sew in observations of missing data from other frames. The resulting data can be used in several applications: we show examples of editing and modifying the data for different animation purposes.

Professor David Forsyth
Dissertation Committee Co-Chair

Professor Jitendra Malik
Dissertation Committee Co-Chair

Contents

List of Figures

Acknowledgments

While a doctoral thesis is billed as the journey of an individual, this process would have been impossible without the help and support of dozens of others. I would like to formally acknowledge some of the people that contributed to this milestone.

The most influential person in this pursuit was my advisor, David Forsyth. He guided my development as a researcher, taught me how to think as a scholar and introduced me to the midwest – a place I would not have explored on my own accord. Thank you David.

Three people contributed directly to the research contained in this thesis: Anthony Lobay, Jai Vasanth and Keenan Crane. I thank all of you for your time and effort — my thesis would not be the same without your help. In addition, I received significant feedback and support from countless other graduate students, spanning most of the vision and graphics groups at Berkeley and Illinois.

Over the years, I've had a number of mentors, and I'd like to acknowledge a few of them here: Mrs. Coleman, my first grade teacher; Mr. Dolyniuk, my middle school math teacher; Mr. Woosnam and Dr. Venuti, my high school physics teachers; Glen George, my instructor and supervisor in college; Pietro Perona, my undergraduate research advisor; and Michael Cox, my manager at NVIDIA.

Finally, I received moral support from my parents, my girlfriend and my sister. Graduate research has long stretches of ups and downs and I want to thank each of you for supporting me when I needed it.

Chapter 1

Introduction

Traditionally, images of cloth are generated in two ways: either by taking photos of real cloth or by simulating and rendering cloth in a computer. These two situations occupy two extremes of a spectrum. Photos look realistic and are relatively cheap and easy to generate but lack the control of simulation. Simulation is not constrained by physics and can be transformed and placed in a new 3D scene, but tends to lack realism. Figure 1.2 shows a pictoral version of the spectrum.

The bulk of our work is somewhere in between: we try to preserve the realistic appearance of actual cloth while giving some degree of control over the scene. We focus on three related projects: retexturing cloth in videos, reconstructing the shape from a single view and reconstructing the shape from multiple views. In the first project, we take an existing image of a cloth shirt and replace the texture. In the spectrum of generating images of cloth proposed in the first paragraph, this is closest to taking photos of actual cloth. The appearance is fairly realistic but the amount of control is minimal. Second, we

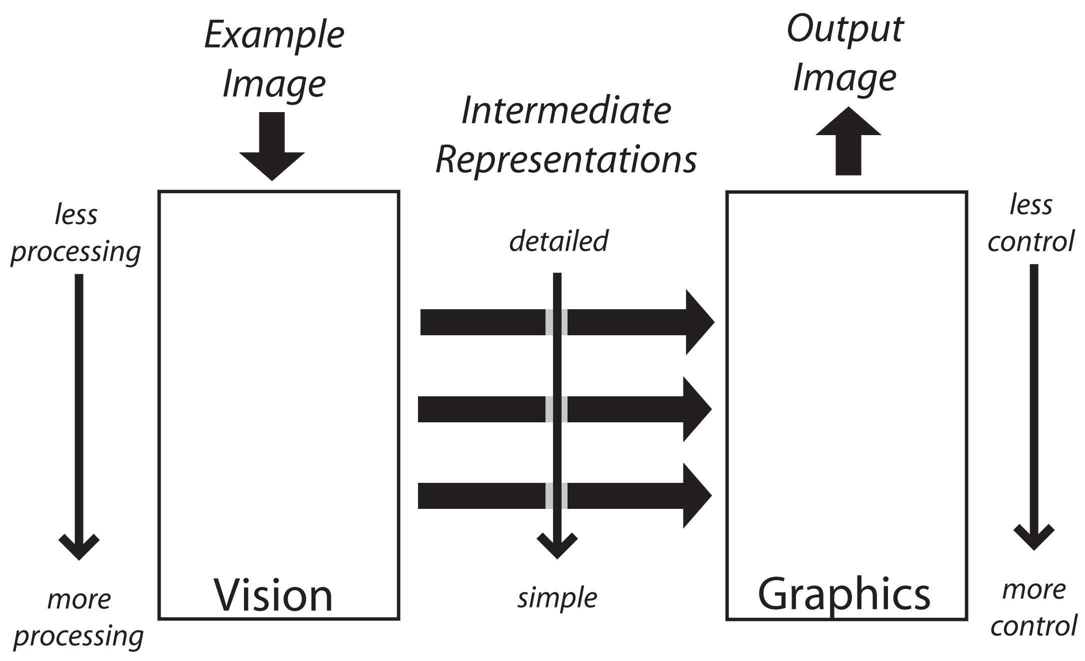


Figure 1.1: In many cases, we wish to use vision to capture parts of the real world and insert it into a virtual world used in computer graphics. In this setting, there are different internal representations that can be used: techniques like *image based rendering* use large amounts of data from the real world and provide realistic virtual environments, but the control over this environment is minimal. In contrast, scanning techniques may acquire 3D mesh models of objects. Typically, the resulting images have lower visual fidelity but allow more control over the environment.

reconstruct the shape of the cloth from a single view. These models are in the middle of the spectrum: they allow some degree of rotation and even relighting, but only cover a small range of views and usually aren't as realistic as the retexturing. Finally, we reconstruct the shape from many views using a custom pattern printed on the cloth. This is the closest to the simulation world: we produce complete 3D meshes that can be viewed from most directions. However, because of the recording setup, they must be re-lit and loose some level of realism.

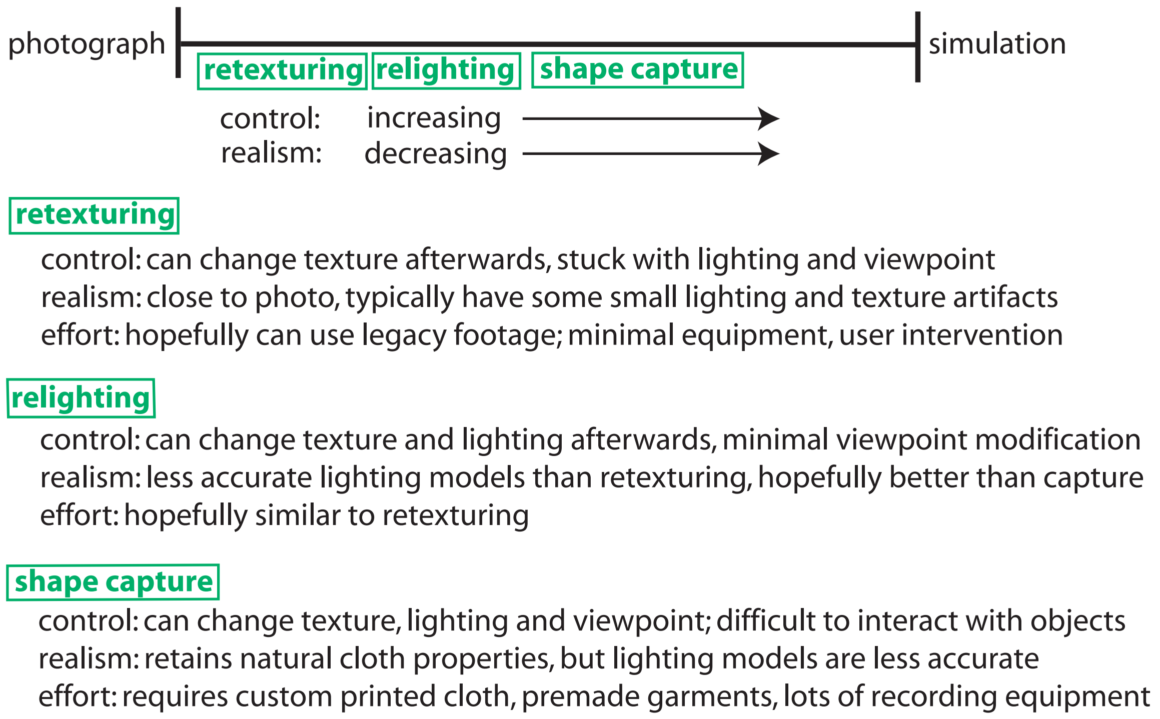

photograph
simulation
retexturing
relighting
shape capture
control: increasing
realism: decreasing
retexturing
control: can change texture afterwards, stuck with lighting and viewpoint
realism: close to photo, typically have some small lighting and texture artifacts
effort: hopefully can use legacy footage; minimal equipment, user intervention
relighting
control: can change texture and lighting afterwards, minimal viewpoint modification
realism: less accurate lighting models than retexturing, hopefully better than capture
effort: hopefully similar to retexturing
shape capture
control: can change texture, lighting and viewpoint; difficult to interact with objects
realism: retains natural cloth properties, but lighting models are less accurate
effort: requires custom printed cloth, premade garments, lots of recording equipment

Figure 1.2: We define a spectrum for producing images of cloth. At one end of the spectrum is photography and the other end is pure simulation. Our work falls in the middle – hopefully giving more realistic images than simulation while allowing some level of control not present in photography.

1.1 Outline

In this thesis, we look at a variety of different techniques to capture and animate cloth that include a variety of different internal representations. This thesis can roughly be divided into three parts: Part I addresses the capture and analysis of cloth in single views (chapters 2, 3, 4) Part II addresses the multiview version of the same problem (chapters 6, 7) and Part III covers mesh techniques to use 3D models of cloth in animation (chapter 8). These sections naturally build on each other: multiview capture of cloth requires many of the same techniques used in the single view version of the problem and the 3D models in

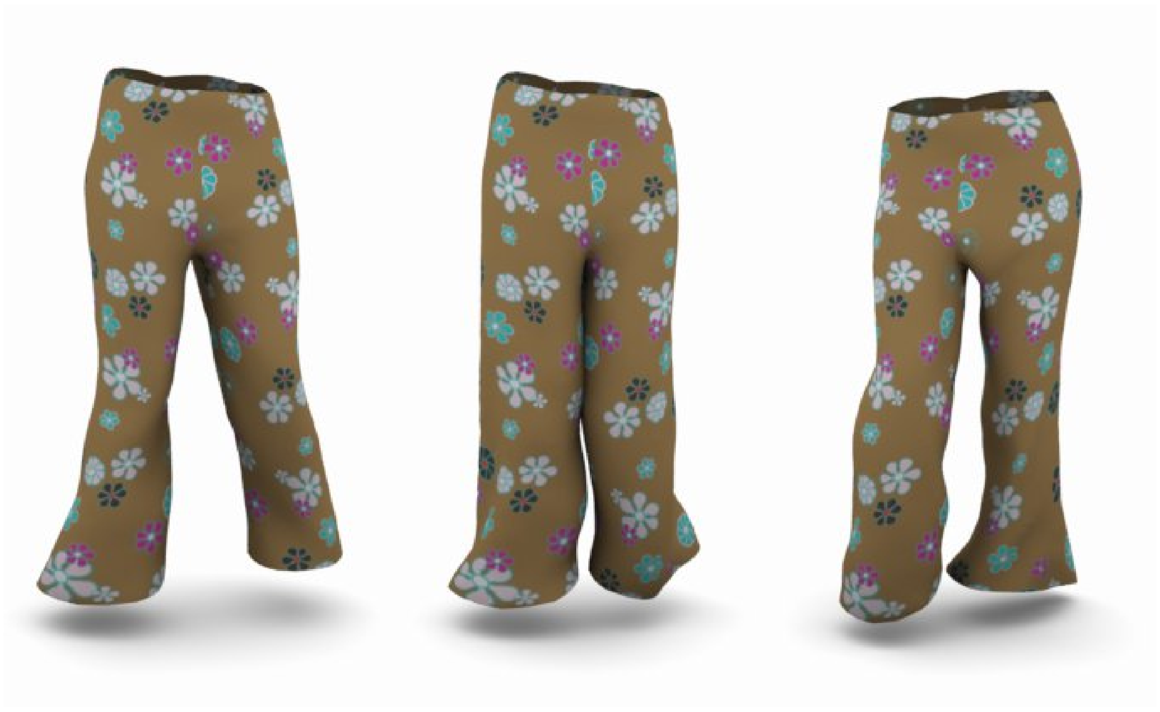


Figure 1.3: This figure shows an example reconstruction: here we recorded a normal pair of pants, then used [Kircher and Garland 06] to add bell-bottoms after the fact.

Part III are acquired using the techniques in Part II.

However, Part I can stand on its own: single view tracking, geometry estimation and retexturing suit a particular set of applications where multiview geometry is either undesirable or even unavailable. The most common circumstance for this is *legacy footage*.

One of the common themes through this thesis is the use of custom printed cloth. Custom printed cloth makes many algorithms faster and more accurate. This is true in both the single view case (chapter 2) and the multiview case (chapter 6). We have elected to include parts of our analysis in both chapters. Chapter 2 will give a general overview of the problem while chapter 6 will discuss specifics in the multiview case and compare to the cloth capture community which is responsible for many of the innovations in the pattern.

Most of the work in this thesis has appeared in a number of other publications. These publications include: *Cloth Capture* [White et al 05], *Deforming Objects Provide Better Camera Calibration* [White and Forsyth 05], *Capturing Real Folds in Cloth* [White et al 06], *Combining Cues: Shape from Shading and Texture* [White and Forsyth 06a], *Retexturing Single Views Using Texture and Shading* [White and Forsyth 06b], *Capturing and Animating Occluded Cloth* [White et al 07].

1.2 General Background

For the most part, we include background in the chapter for which it is relevant. However, some topics, such as cloth simulation, are relevant to the entire manuscript.

1.2.1 Cloth Simulation

There is a substantial literature on cloth modelling; only a superficial introduction is possible in space available. Cloth is difficult to model for a variety of reasons. It is much more resistant to stretch than to bend: this means that dynamical models result in stiff differential equations (for example, see [Baraff and Witkin 98, Terzopolous et al 87]; the currently most sophisticated integration strategy is [Bridson et al 03]) and that it buckles in fine scale, complex folds (for example, see [Bridson et al 02]). Stiff differential equations result in either relatively small time steps — making the simulation slow — or in relatively heavy damping — making the cloth slow-moving and "dead" in appearance. Cloth has complex interactions: it collides with itself and rigid objects; it is driven by forces that are hard to model, including human motion and aerodynamics. Collisions create difficulties because the fine scale structure tends to require large, complex meshes, and resolving collisions can be tricky; for example, careless resolution of collisions can introduce small stretches (equivalently, large increments in potential energy) and so make a simulation unstable (for example, see [Baraff et al 03]). A summary of the recent state of the art appears in [House and Breen 00]. While each of these issues can be controlled sufficiently to produce plausible looking simulations of cloth, the process remains tricky, particularly for light, strong cloth (e.g. woven silk), where the difficulties are most pronounced.

Part I

Single View

Chapter 2

Texture Coordinates in a Single View

When recording and representing the state of an image of cloth, one of the most important aspects is the texture on the cloth. We need to compute a set of texture coordinates (or, equivalently, material coordinates) from our observed image. We use two possible strategies: each has advantages and disadvantages. First, we could engineer a pattern that worked like a map (section 2.2); it would be easy to determine where a particular color lies in the map, and so we could use the color classifier outputs to determine which point on the map corresponds to a particular pixel in the image. Second, we could work with arbitrary textures. In this case, we assume that the cloth in the video has some texture that reveals the structure of the cloth. The net result of these activities is that we can estimate texture coordinates in many circumstances. There are two prices: less precise estimates of texture coordinates and a more costly matching system. In this method, we require a frontal view of the cloth in order to estimate the texture coordinates. As shown in section 2.4, we can actually estimate this texture from video.

2.1 Background

Texture Replacement: Texture replacement falls into two categories: replacing large amounts of texture by placing a custom pattern on a shirt or replacing repeated texture on a textureless surface.

We focus on two papers that retexturing using custom patterns: [Bradley et al 05] and [Scholz and Magnor 06]. In [Bradley et al 05], the 2D texture transformation is estimated using the custom pattern of dots with unique texture around each dot. Most of the pattern is the background color (white) and the small bits of black are replaced by interpolation. The work of [Scholz and Magnor 06] is similar to our own: they encode the texture transformation using colored markers. Lighting is estimated from nearby background regions (which are unfortunately dark in color) and then interpolated inside each marker. [Scholz and Magnor 06] consider the more challenging case of a pattern that covers the entire pattern. As such, they focus on the occlusion and boundary problems. They estimate silhouettes and occlusion masks to compute the correct replacement.

Textureless surface tracking and replacement papers include [Fang and Hart 04], [Zelinka et al 05], and [Fang and Hart 06]. They use a coarse shape from shading estimate to lay down the repeated texture.

Tracking: Several methods have been proposed to track nonrigid motion including [Tsap et al 00] and [Pilet et al 05]. [Pilet et al 05] describe a method to detect the surface deformation using wide-baseline matches between a frontal view and the image to estimate a transformation smoothed using surface energy. This method cannot stabilize texture for three reasons. First, there are few reliable keypoints in the texture we consider, especially in oblique views. Second, by using keypoints, the method does not track boundaries — and oscillations in the boundary conditions are noticeable in their videos. Third, the rigid smoothing using surface energy makes their method stiff, and limited in scope. In addition, they cannot obtain an irradiance estimate — small errors in their correspondence would make it impossible.

[Lin 05, Lin and Liu 06a] track near-regular deforming textured surfaces using a lattice-based Markov random field model that effectively assumes spring-like movements. In both cases, energy terms are used to keep the tracker in the right position.

2.2 Printing a Texture

First, we look at the problem of getting texture coordinates from a custom printed texture. In this scenario, we print a texture onto the surface before filming the sequence. This is more restrictive (since we can't alter legacy footage), but it makes our task easier and more accurate.

In this chapter, the goal of printing a texture is to make it fast and easy to extract texture coordinates. We will encode texture coordinates as a transformation between an undeformed version of the cloth and the observed one. This encoding is natural for texture replacement: to put on a new texture, we just need to run it through the same transformation.

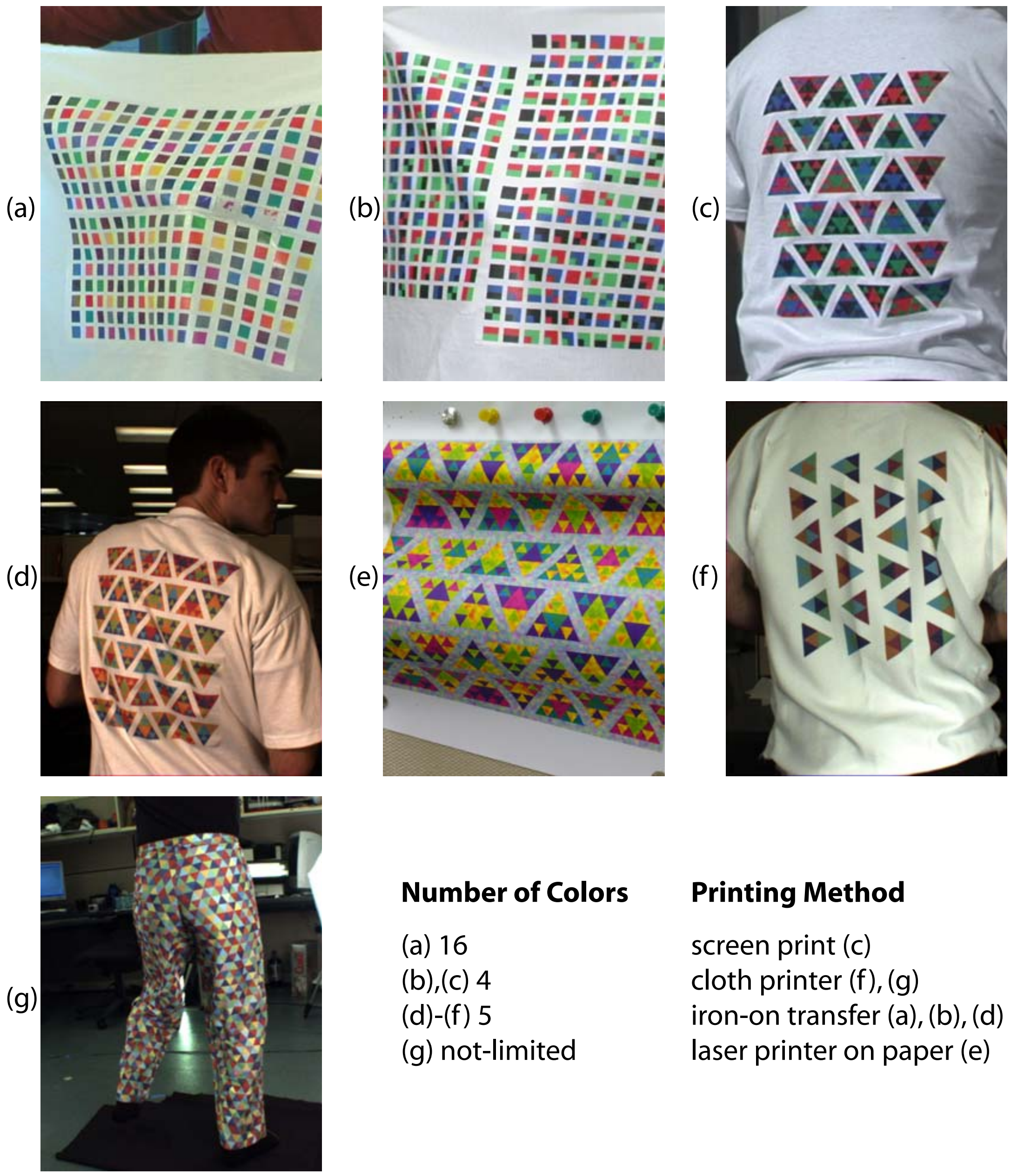
For cloth capture, color is the marker descriptor of choice: colored markers make disambiguation substantially easier without eating resolution. Color cameras are common and typically the same price as black and white cameras. The downside is that for many cameras the Bayer pattern degrades the image quality slightly.

We use large colored regions with constant color. These large regions are beneficial because we can average over them to get an accurate color sample and they are robust to motion blur: the center of the region is rarely affected by motion blur enough to make detection difficult.

Internal Structure

We considered two ways to encode information on the surface of the cloth: information that is encoded at a single scale (such as a tessellation of markers of different colors) and one that includes information at multiple scales (where larger markers include enough information to determine exact location on the cloth). Both of these methods have been used by other authors as well [Guskov et al 03], [Scholz et al 05] and [Scholz and Magnor 06].

Information at one scale is sufficient because these patterns implicitly encode information at multiple scales without the occlusion drawbacks of explicitly multiscale techniques. If part of the large scale element is occluded, it's identity can't be decoded and all of the smaller elements are lost. However, a detection algorithm that works at the smallest scale doesn't fail when large scale structure is occluded. Since textures with large scale structure offer no advantages over those with information exclusively at the smallest scale, we opt for the small scale texture (see section 2.2.1).



Number of Colors	**Printing Method**
(a) 16	screen print (c)
(b),(c) 4	cloth printer (f), (g)
(d)-(f) 5	iron-on transfer (a), (b), (d)
(g) not-limited	laser printer on paper (e)

Figure 2.1: We have experimented with a large number of different patterns. We identify the following distinctions in pattern types: (1) number of colors, (2) inclusion of whitespace, (3) addition of noise, and (4) internal structure. Different tasks force different requirements, however we believe there are common themes. First, internal structure isn't necessary — random patterns are actually easier to work with. Second, whitespace has few advantages and consumes valuable pixels (for a discussion on effective use of pixels, see figure 5.2). Third, if a reasonable color model can be established more colors are typically better.

There are a number of additional design attributes for creating custom patterns which we address below.

How Many Colors? What Colors?

In general, the more colors that can be distinguished the better. However, as the number of colors printed on the surface grows, so does the potential confusion between similar colors. One approach would be to print the largest number of colors tailored to the particular camera and lighting setup. This, however, is challenging: the color changes dues to printing, lighting and recording are difficult to calibrate. As a result, most techniques use four or five colors to maintain robustness.

We have experimented with as many as eight distinct colors successfully, but the color calibration efforts become increasingly challenging. As a result, when many markers are involved, we advocate a random color pattern and a statistical model for detection. In later chapters, we show that this pattern can easily achieve the equivalent of 32 colors – far more than any palette based method.

Include Noise?

Our basic approach has been to use markers of constant color in order to be robust to imaging noise, shadows and motion blur. However, we pay a cost for this robustness: we have difficulty resolving fine scale structure. If we were to print noise into the pattern, we would be able to resolve more detail: especially in the multiview case where we could use space carving to estimate the geometry. However, we did not have the time to extensively test noise in the pattern.

Include Whitespace?

Finally, some patterns (such as [Scholz et al 05]) include whitespace in the pattern. For direct cloth capture, this is inappropriate: whitespace means that there is less information per unit area on the cloth and thus less geometry is recovered. However, whitespace makes lighting much easier to estimate. We believe that this is unnecessary, lighting can be estimated from colored regions as long as the colors are relatively light.

2.2.1 Tracking Structure at Only One Scale

For markers at a single scale, we find blobs within the image, estimate the spatial extent of the blob, determine the color and observe the neighborhood structure. Each frame is treated independently from the next. To compute correspondence between a view of the folded cloth and the frontal domain, see Chapter 6.

We start by converting each image to HSV, disregarding the luminosity (V) and using polar coordinates to compute distances in hue and saturation. To detect markers, our code looks for uniformly colored blobs in two stages: first regions are built by growing neighborhoods based on similarity between pixels. This method is sensitive to image noise and can produce oversized regions when the color boundaries are smoothed. The second stage takes the center of mass of each blob from the first stage, computes the mean color and grows a region based on distance to the mean color (it is computationally intractable to use this as the first stage of the blob detection). The process is iterated for increasing thresholds on the affinity value in the first stage, using the portions of the image where detection failed in previous stages. Finally, blobs are thresholded based on size.

Next, we need to determine the neighborhood relationships. For each marker, we construct a covariate neighborhood (a fitted ellipse) and vote for links to the three closest markers with similar covariate neighborhoods. This measures distances appropriately in parts of the scene where the cloth is receding from view and discourages links between markers with wildly different tilts. All links that receive two votes (one from either side) are kept while the rest are discarded. Links that bridge markers with conflicting color information are also discarded (typically on internal silhouettes).

2.2.2 Tracking Structure at Multiple Scales

This section is for reference only: it discusses what we did to detect structure at multiple scales for historical and explanatory purposes. Many of the figures in this thesis were generated using this underlying method, even though better methods are now available (see the previous section).

Here, the texture is a set of large equilateral color coded triangles where the coloring of each triangle identifies the location and orientation on the cloth. Each large triangle consists of a number of small triangles – where the vertices of the small triangles form a fine grid like structure over the cloth (figure 2.2). First, elements are highly distinctive and there is little repetition over a large area of cloth. If the cloth is moving quickly, some cameras may see only a small fraction of the entire cloth, so that global correspondence reasoning is impractical. A distinctive element allows reconstruction even in this difficult case. Second, our pattern offers a high degree of spatial accuracy, while allowing robust observations during dynamic sequences. These two requirements are in tension because motion blur tends to obscure the high frequency information need for accurate localization. Our large triangles are relatively easy to identify despite motion blur; a deformable template approach then yields the interior structure (15 vertices of the smaller triangles), in a form of coarse-to-fine search.

Coarse step — finding large triangles and their normals: We assume that, over the scale of an individual triangle, perspective effects are negligible; that over the scale of the whole frame, the effects of perspective are small; and, for the purposes of normal estimation, that the surface curvature at the scale of a triangle is small. We binarize the image by thresholding based on average color values. Using morphological operations, we compute a list of all the elements in all frames from a single camera. Since our pattern is highly repeated, we can then use a histogram to determine the rough size of the triangles. Because perspective is negligible at the scale of an element, each triangle is imaged through an affine transformation that is a function of camera scale, the slant and tilt of the plane on which the triangle lies, and the in-plane rotation of the triangle. We now obtain a rough estimate of camera scale by assuming some triangles in the sequence will be viewed frontally by this camera, so that the largest triangles offer an estimate of camera scale. We now use the scale estimate to precompute views of each triangle at a set of different slants, tilts and in-plane rotations. Then, for all observed triangles we perform a quick comparison using sum-of-squared differences to pick the closest precomputed triangle as a start point for a continuous optimization over slant, tilt and in-plane rotation (we have found no practical need to include camera scale and triangle position in this optimization). The result gives the normal at the large triangle, whose location is known as above.

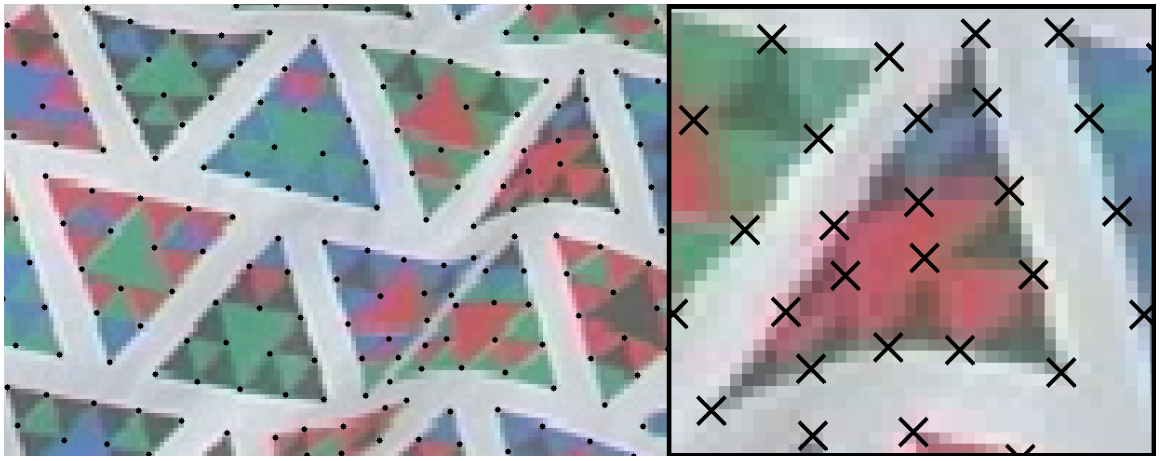


Figure 2.2: This image shows one possible pattern that we can print on the surface of the cloth. It contains information at multiple scales: the larger triangles have a color coded pattern that defines correspondence between views; the vertices of the smaller triangles provide large numbers of point correspondences. As it turns out, the coarse to fine nature of this pattern isn't all that useful and the pattern is less robust to the more difficult and common problem of self occlusion. We extract this information in a coarse to fine search. Because we know the frontal pattern of the cloth and because we assume that there are few folds smaller than the scale of the smallest triangles, we can compute normals to each of the small triangles. The reprojection error in figure 7.2 confirms the validity of this assumption.

Fine step — localizing triangle vertices: For each large triangle, there are 15 triangle vertices. To localize these vertices, we need to compare the actual structure of the large triangle with the image, which requires color comparisons. Image color is a surprisingly poor guide to object color, as it is affected by shadows, variations in printing, lighting and camera sensitivities. Because we know the location of the large triangles, we can rectify the color with a simple strategy. We know that the color pixels are distributed uniformly amongst red, green, blue and black. We then allocate pixels to colors using a greedy strategy, rather like round-robin: assign the red most pixel the label red, the green most green, et cetera, then repeat until no pixels are left.

We now localize each point using a deformable model. Starting with the initial

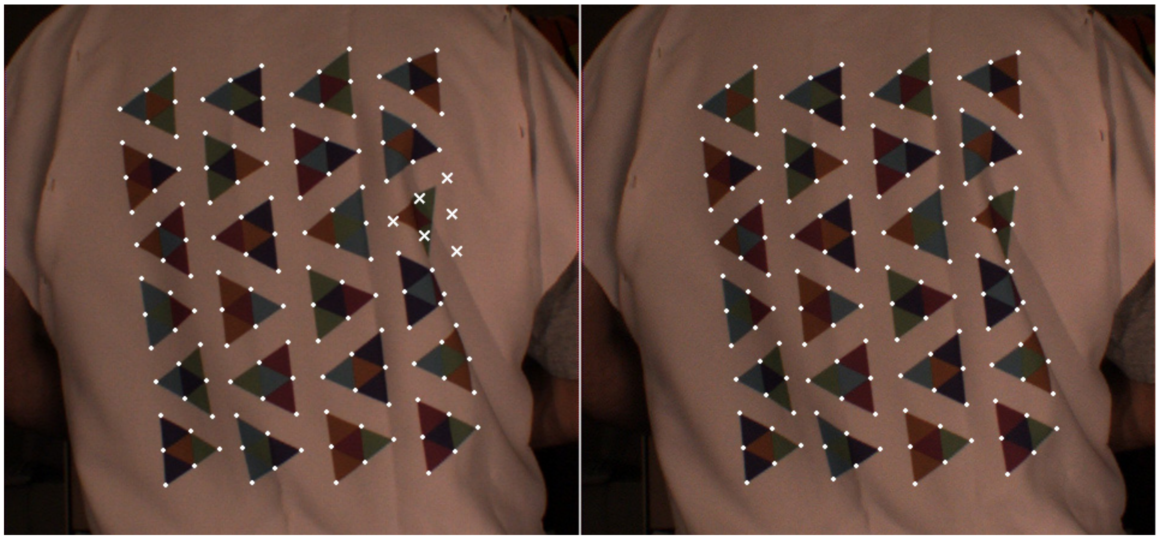


Figure 2.3: Here we see one of the problems with a top down method for marker identification: large regions go missing with small changes in geometry. On the **left** a single triangle is not detected. In this case, we interpolate the location of the triangle using a linear estimate based on the locations of the other triangles (shown as white Xs). In the next frame (**right**), the corresponding triangle is detected with a substantially different position — causing a pop in the video sequence.

estimate of the large triangle configuration, we allow a model of the triangle to deform – using the vertices of the smaller triangles as free variables. Charging for differences between the modeled triangle and the actual triangle, we use numerical optimization to find the optimal match. Even in cases where the colors are blurred and corners are not visible, because this method uses the large regions of uniform color to drive the optimization, it can produce reasonable estimates of the vertex locations.

2.2.3 Printing Technologies

As shown in figure 2.1, there are a number of different ways to print on cloth. This section is likely to become outdated, however it may still be useful as a historical reference in the future. He tried three basic printing technologies: screen printing, iron-on transfers and digital printing on cloth surfaces. We evaluate printing technologies by the quality of the colors, the control over the appearance of the cloth, the effect on the dynamics of the cloth and the associated costs. For cloth capture, digital printing is superior.

Screen printing is one of the more common traditional methods to print graphics on cloth. In this technique, ink is pressed through a screen to the surface of the cloth. The screen is a fine mesh that can be chemically plugged. In regions where the screen is plugged, ink is not printed. In regions where the screen is not plugged, ink is applied directly to the fabric surface.

Screen printing is common because it is a cheap way of making large number of identically colored garments. Raw materials ink are relatively cheap, and printing is fast once the screen is made. However, making the screen is somewhat time consuming, and at the time of this thesis runs about a hundred dollars. A screen must be made for each of the colors on the surface. As a result, few screen print items have more than 3 or 4 colors. Because the ink is applied directly to the surface, typically the layer of ink is somewhat thick – slightly altering the dynamics of the surface. However, the resulting colors can be heavily saturated.

Iron on transfers are thin sheets of material that fuse to the surface of the cloth when heated. One can print directly onto the thin sheet of material using standard printers (such as a laser printer). As a result, any range of colors is possible and simple computer tools make it easy to create and alter patterns. Typically the resolution of these techniques is higher than screen printing. However, iron-on-transfers significantly alter the dynamics of the cloth and are undesirable for cloth capture as a result.

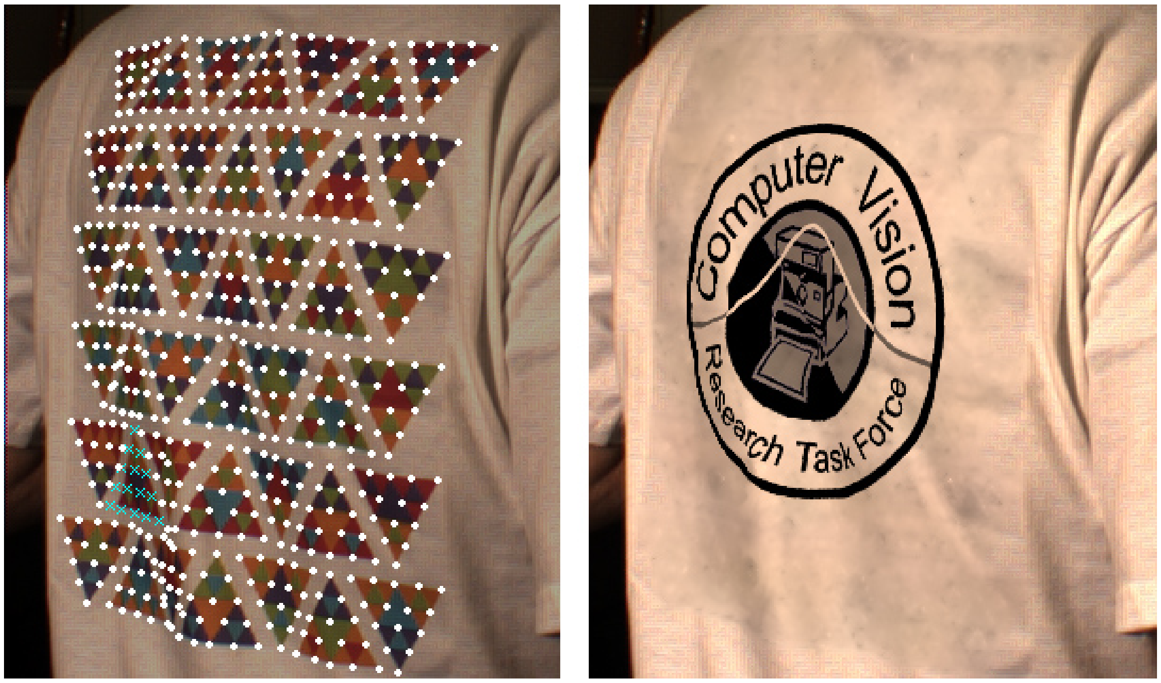


Figure 2.4: Another example of detection on a texture with information at multiple scales. Here, the original texture is put on the cloth using a technique known as *iron-on transfers*. This is inferior to fabric printing: the characteristics of the cloth change as a result of the new material that adheres to the original cloth. As a result, a sharp crease appears at the edge of the transfer and the dynamics are substantially altered.

Digital printing involves a printer and ink that directly prints on the surface of the cloth. These printers are expensive and the colors are less saturated in general, but the cloth dynamics are almost un-altered by the ink. At the time of writing, www.silviascostumes.com and dyo.customink.com provide digital printing services: Silvias costumes prints onto fabric by the yard while Dyo Customink prints directly onto t-shirts.

2.3 Natural Patterns

While the pattern detection approach in Section 2.2 is compelling, it is somewhat specific to the custom printed pattern. For arbitrary screen print textures, localization becomes a problem. Instead, we adopt a top-down method to fit the texture: first, search for the rough shape (Figure 2.5) then refine the mapping (Figure 2.6). We use a triangle mesh to represent the mapping, splitting triangles as the mapping is refined.

This method has several advantages. First, no region of the pattern needs to be particularly discriminative; only the pattern as a whole has to be unique. Second, highly oblique views still exhibit the overall shape and can be detected. Third, edges are powerful cues in this model: they provide a constraint that is not easily recorded using point feature correspondences. Fourth, in contrast to surface energy methods, this method does not have many of the common stiffness properties. However, the disadvantages of a top-down approach are not insignificant: it is not robust to occlusions and subject to local minima. Practically, this means that partial views of the surface may not be retextured properly.

Estimating an Initial Correspondence: Our method of fitting proceeds in two steps: first, estimate the rough location and scale of the logo, then refine the estimate. Because the quantized image has fewer lighting effects, both stages are performed on the output of our color classifier, not the original image. To detect the rough location of the object we use a color histogram with 16 spatial bins (arranged in a 4×4 grid) and the same number of color bins as colors in the texture, resulting in a histogram of size $4 \times 4 \times \mathcal{C}$. Following other work with histograms [Lowe 04], we normalize the values, suppress values above 0.2, then re-normalize. Using the descriptor from the frontal image as a query, we perform a combinatorial search over scale, location and aspect ratio in a downsampled version of the target image.

Refining the Transformation: Once the rough transformation has been com-

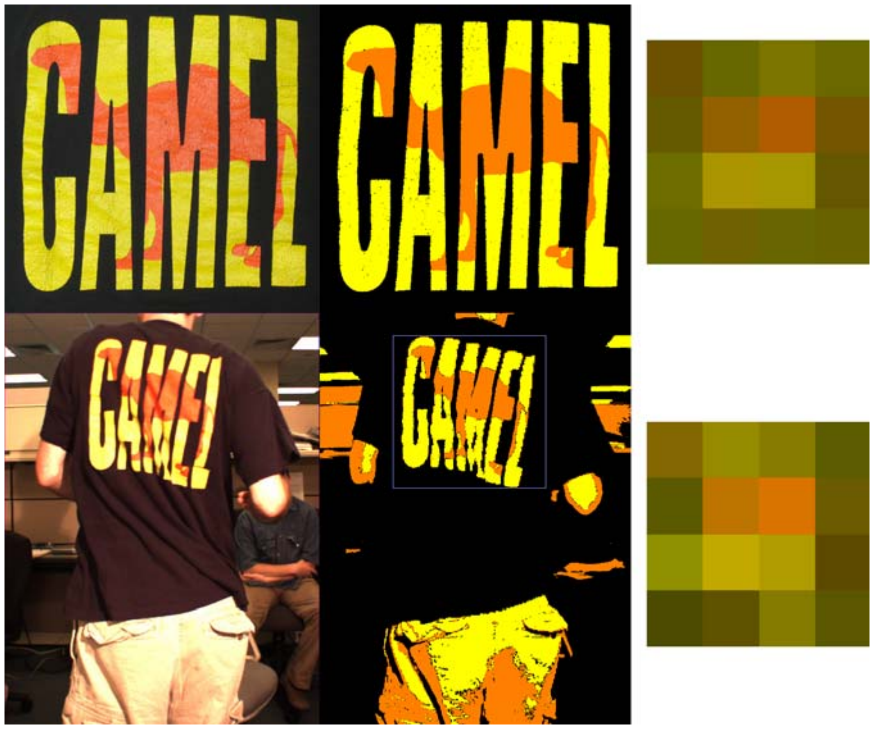


Figure 2.5: Our method requires a frontal view of the texture (**top row**) and a target image (**bottom row**). We quantize the colors using a color classifier, then compute a 4×4 color histogram, with separate bins for each color channel. In this case, with three colors (black, orange and yellow), our descriptor is 4×4×3. We visualize this descriptor by reconstructing the colors and normalizing appropriately (**right**). A search for the descriptor in a subsampled version of the target image reveals the closest match (**bottom right**).

puted, we refine over scales (Figure 2.6). At each stage in the refinement, we implement the same algorithm: blur the color quantized image (giving each quantized color its own channel), then run gradient descent over the locations of the vertices using the sum of squared distances between the transformed frontal texture and the blurred target image. Our model of the transformation is coarse: we start with a 4 vertex 2 triangle model, then refine to 9 vertices and 8 triangles, and finally 13 vertices and 16 triangles.

This representation has several advantages: first, even coarse textures without many distinctive points can be localized. Second, we can refine the transformation as

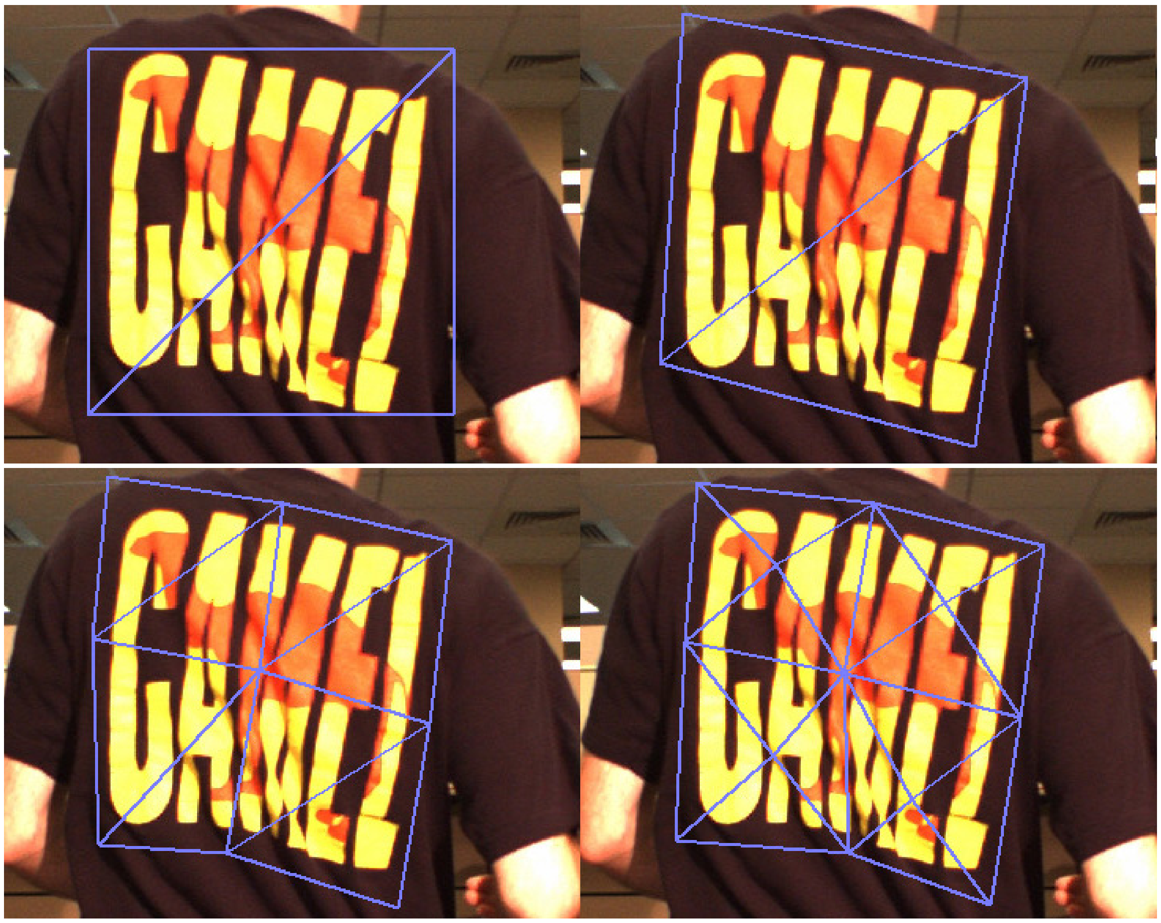


Figure 2.6: Our method refines estimates of texture location over scale. Starting with the output of initialization step (figure 2.5), we have an axis aligned box that corresponds roughly to the location. We use gradient descent on blurred versions of the color quantized image to improve the transformation. Iteratively, we refine the number of vertices (and correspondingly the number of triangles) while reducing the blur to get a better match. Our final model contains only 16 triangles.

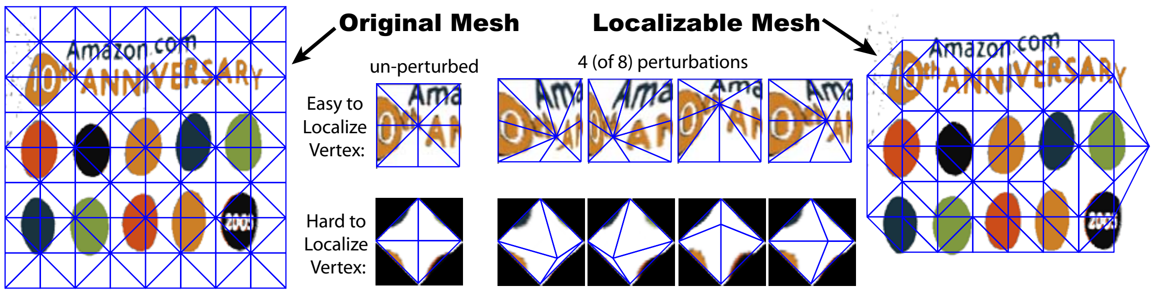


Figure 2.7: We generate fine scale meshes by subdividing a coarse mesh, then measure the localizability of the mesh vertices — pruning points that cannot be easily localized. A point is localizable if a perturbation of the location forces a significant change in the transformed image. After throwing away points that are not localizable, we compute a delaunay triangulation of the resulting points and throw away triangles that are ill-conditioned to get a final mesh.

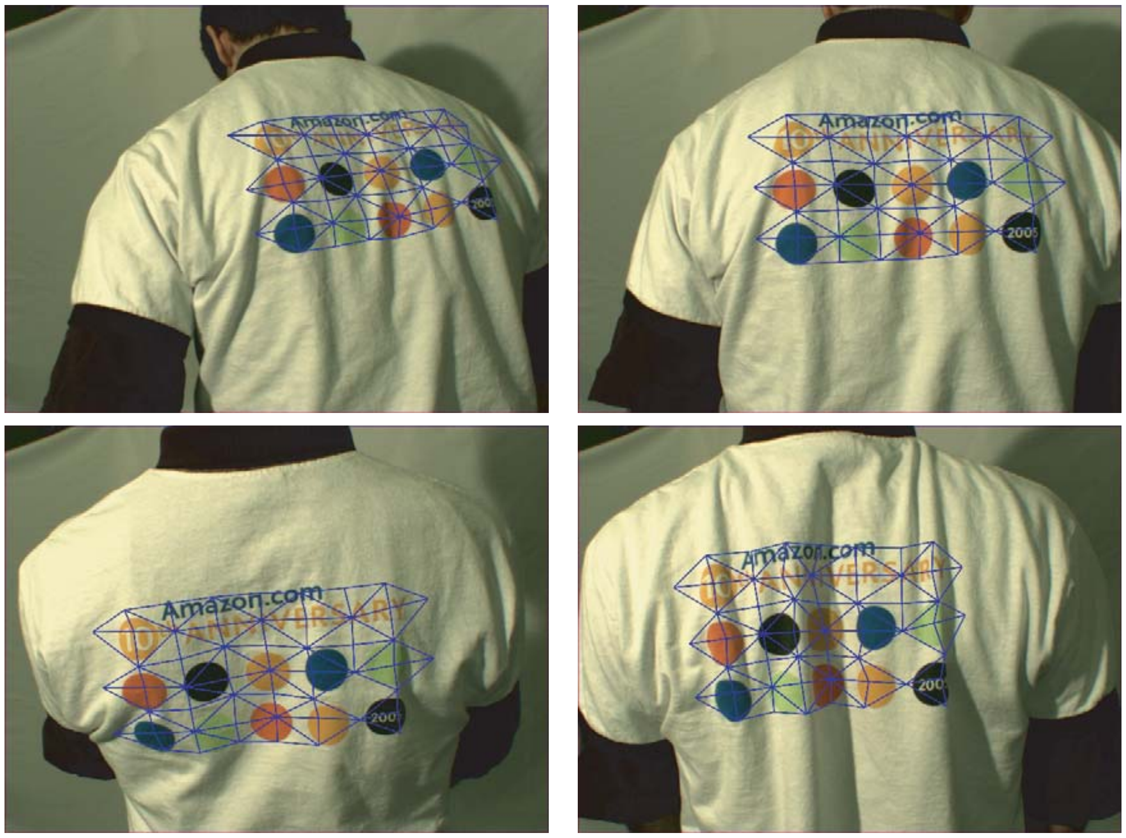


Figure 2.8: We show some tracking results.

appropriate. At some point, continued refinement is no longer appropriate: the surface texture may not have localizable texture below a certain scale. (Figure 2.7)

2.4 Estimating Texture from Video

In general, a frontal view of a texture is required to compute the surface normals in an arbitrary image. Forsyth [Forsyth 02] considered the case of a single view of a repeating pattern, and showed that 3 views of the repeating pattern (assuming no degeneracies) are enough to determine the frontal view. This method assumes that texture elements are small and therefore approximately planar. We extend this notion to larger textures with repetition in time instead of repetition in space. By observing a video of a moving *non-rigid* texture, we estimate the frontal view by assuming that small regions of the texture are roughly flat. Stitching together the estimated frontal appearance of each small region, we get the frontal appearance of the entire texture.

As a result, we can reconstruct the shape of a deforming surface from a single video. We only require that the user select the texture to track by clicking four points in a single frame of the sequence. (At present, we also require the user to individually select the colors of the screen print pattern in order to build a classifier [White and Forsyth 06b])

Because we operate under the same assumptions as Forsyth, this procedure requires only 3 views of the deforming surface. However, in practice we typically need more — a degeneracy in any triangle can cause problems in the resulting mesh. In figure 2.9 we estimate the frontal appearance from 8 views of 128 triangles because no set of three views lacked degeneracies. However, in dynamic video sequences a short segment should be

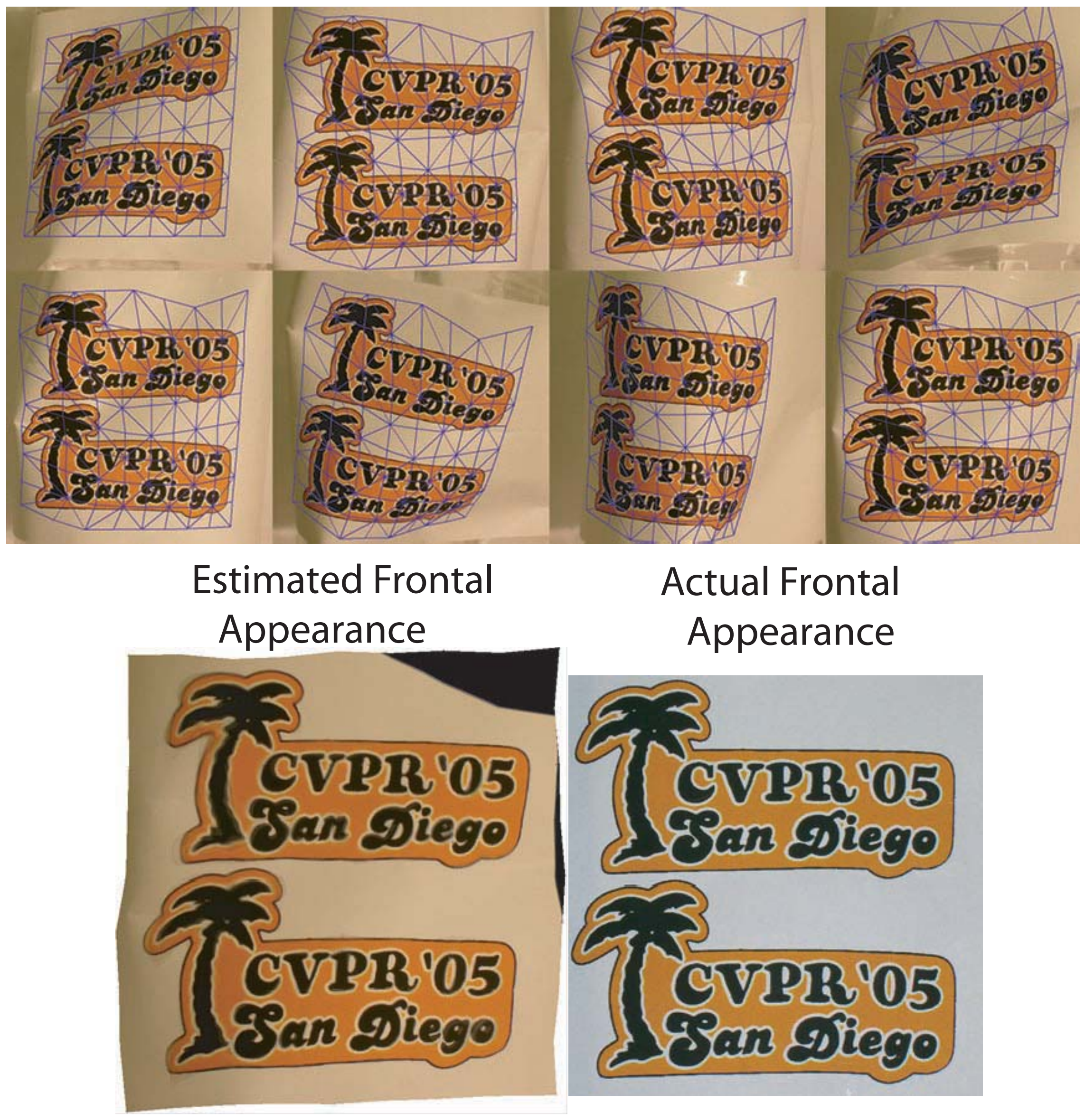


Figure 2.9: We automatically compute the frontal appearance of a deforming surface — allowing us to compute normals and thus estimate shape (figure 4.6). The estimated frontal texture is an average of the eight images, each pushed through the appropriate warping. We require the user to manually select the four corners of the texture in a single view (in this case, the upper right image). Then we automatically deform this model to find matches in each of the other frames (a total of eight in this case). The blue mesh in each of the images on the left indicates the match. Following the method outlined in the text, we compute estimated frontal edge lengths, then, treating these lengths as rest-lengths for springs, we minimize energy. Without assuming any form of surface rigidity, this method requires three views of the deforming surface. However, because of degeneracies that arise in practice, we use eight. Degenerate triangles occur when there is a single axis of rotation between views — detecting these degeneracies is described in section 2.4.1.

sufficient.

2.4.1 Implementing Frontal Estimation

Assuming that we have a mesh of static connectivity that tracks the surface over a sequence of frames, we treat each triangle separately. A 3D reconstruction can be computed (with some sign ambiguities) assuming orthography from 3 planar points [Huang and Lee 89, Ullman 79]. We extract sets of three triangles, compute a 3D reconstruction, check the reconstruction for degeneracies by computing the angles between cameras and add the lengths of each edge to separate running edge length lists. After iterating over sets of three views of each triangle and iterating over triangles in the mesh, we have a list of estimated edge lengths for each edge in the mesh. We pick the median value, then minimize a spring energy term to confine the points to a plane.

Chapter 3

Shading on Screen Print Surfaces

Careful estimates of irradiance are useful, and appear to create a powerful impression of shape. Their significance for renderings of clothing is probably due to *vignetting*, an effect which occurs when a surface sees less light than it could because other surfaces obstruct the view. The most significant form for our purposes occurs locally, at the bottom of gutters where most incoming light is blocked by the sides of the gutters. This effect appears commonly on clothing and is quite distinctive [Haddon and Forsyth 97, Haddon and Forsyth 98]. It is due to small folds in the cloth forming gutters and shadows and could not be represented with a parametric irradiance model unless one had a highly detailed normal map.

However, we do not have and cannot get a detailed normal map or depth map. Furthermore, the estimates of material coordinates may be of limited accuracy. As a result, irradiance estimates that use the material coordinates are inaccurate, especially in regions where the albedo changes quickly. A single pixel error in position on the texture map can,

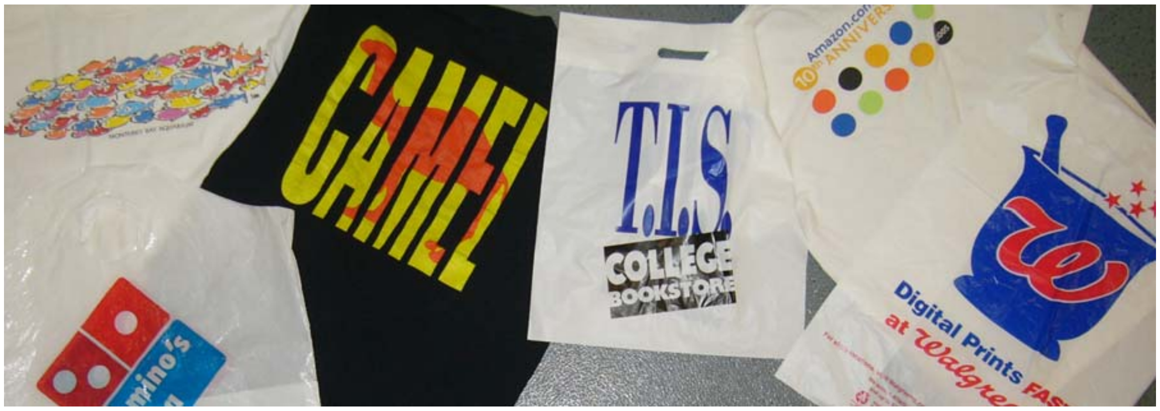


Figure 3.1: Many common textured articles are made using *screen printing* — where each color is printed in a separate pass. Often, this method is cheaper than using a full color gamut. Screen printing is widespread: many T-shirts, advertisements, and corporate logos are composed of a small number of solid colors. Recovering irradiance is easier in this setting: correspondence to a frontal view of the pattern is not required. Instead, each color can be detected independently in order to recover irradiance. Because screen print items are composed of large regions of uniform color, they are robust to motion blur.

for example, mean that an image location produced by a dark patch on the shirt is ascribed to a light patch on the shirt resulting in a catastrophically inaccurate irradiance estimate. These errors probably explain why correspondence tracking methods ([Pilet et al 05]) do not estimate irradiance or use it to retexture — the correspondence is not pixel accurate, meaning that irradiance estimation would probably fail.

What we do have is an assumption that the clothing pattern is screen-printed, using a small set of highly colored dyes in regions of constant color. In this case, we do not need a formal estimate of irradiance. Instead, at any point in the image, we need an estimate of what the reference (background) color would look like, if it appeared at this point. By taking this view, we avoid difficulties with scaling between pixel values and radiance, for example. We can obtain this estimate in three steps. First, we build a table that indicates, for each of the dyes in the screen print, what the reference color looks like in illumination

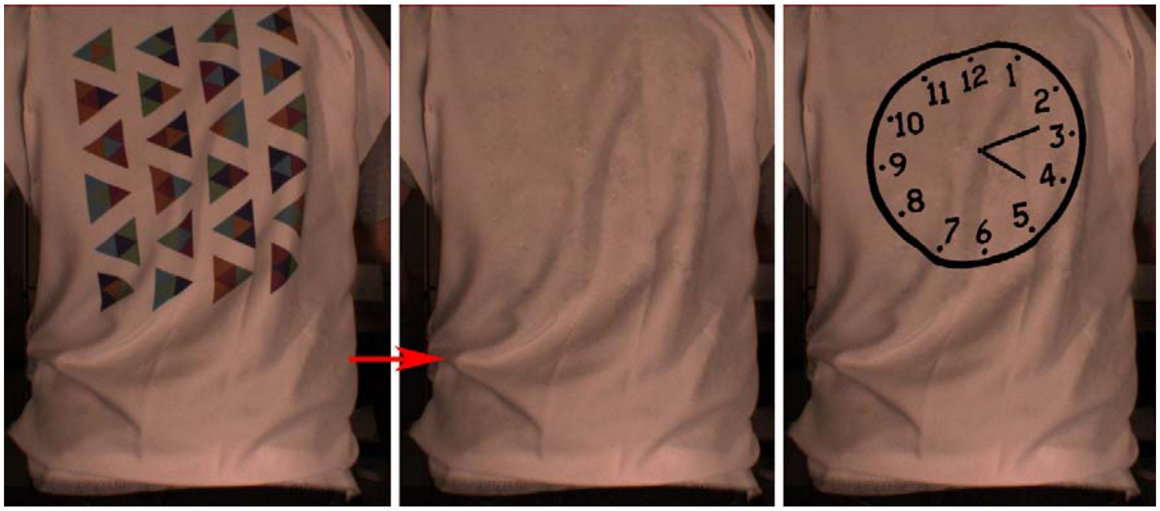


Figure 3.2: Lighting cues provide a strong sense of shape — with or without a new texture. **Left**, an image from a video sequence taken directly from our video camera. **Middle**, we remove the texture items by estimating the irradiance and smoothing. **Right**, a retextured image. Notice that irradiance estimates capture shape at two scales: the large folds in the cloth that go through the middle (starting at the arrow, follow the fold up and to the right) and the finer creases.

that produces a given set of image R, G and B values from the dye. Second, at each image pixel, we determine what (if any) dye is present and use the look-up table to estimate the appearance of the reference color at that point. Third, we smooth the resulting field.

3.1 Regressing the effects of irradiance

We do not require irradiance: It is sufficient to know what a white patch would look like when a given dye patch has an observed appearance. This information can be obtained by regressing from observations. We build one table for each dye, using the following approach. We use our color classifier (below) to identify pixels from that dye that lie next to pixels produced by white patches. It is reasonable to assume that, if the pixels are sufficiently close, they undergo the same irradiance. We now have a series of

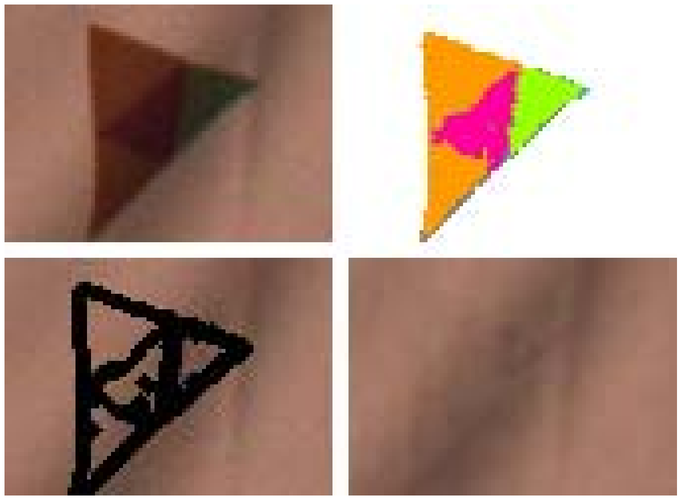


Figure 3.3: We estimate lighting for each dye independently, throwing away confusing pixels and boundary regions. **Upper left**, an un-altered image of a triangle in our pattern contains strong lighting cues. However, the boundary regions yield conflicting cues: boundary colors change in somewhat unpredictable ways, hiding the strong lighting cues. **Upper right**, the output of our color classifier, run on a per pixel basis. As noted, colors at edges can be confusing (indicated in gray) or misclassified (notice the blue pixels at the right tip). **Lower left**, after eroding each colored region, we extract lighting cues for each pixel independently. At this stage, there are two problems in our lighting model: gaps in the irradiance estimates and slight chromatic aberrations. In the **lower right**, we interpolate regions using a Gaussian of varying width. To smooth out chromatic aberrations, we convert to HSV and heavily smooth both hue and saturation.

examples, linking image RGB of the dye to image RGB of white. The number of examples is enormous; one might have 10^5 or even 10^6 pixel pairs in a given video. However, some examples may be inconsistent, and some image RGB values may have no entry.

We obtain a consistent entry for each image RGB value that occurs by identifying the mode of the examples. We now have a table with some missing entries (where there were no examples). We use a version of Parzen windows to smooth this table by interpolation.

3.2 What dye is present?

We determine which dye is present with a classifier that quantizes the color of each pixel to a pre-determined finite set (determined by the user) based on the pixel's component colors. The classifier is a set of one-vs-all logistic regressions on first and second order powers of RGB and HSV. To classify pixels, the maximal response from the array of classifiers is selected, except when all classifiers respond weakly, in which case the pixel is labeled as ambiguous. We do not attempt to classify pixels close to color boundaries, because blur effects in the camera can lead to classifier errors. At present, we require the user to click on each color to train the classifier, but believe that clustering could remove this step of user intervention.

3.3 Interpolating, Smoothing, and Blending

We now take the pool of relevant pixels, determine what dye is present and do a dye specific table lookup using the RGB values as indices. The result is a representation of what the image would look like at that pixel if the dye had been white. However, this is not available at every pixel — the classifier might refuse to classify or the pixel might be close to a color boundary and dangerous to classify. Missing pixels are interpolated using a Gaussian weight to sum up nearby pixels, with the variance corresponding to the distance to the nearest pixel. Our irradiance estimates often have slight errors in color. Observing that color variations in lighting tend to be low frequency, we heavily smooth the hue and saturation of the recovered irradiance. (Figure 3.3). Finally, using the domain of

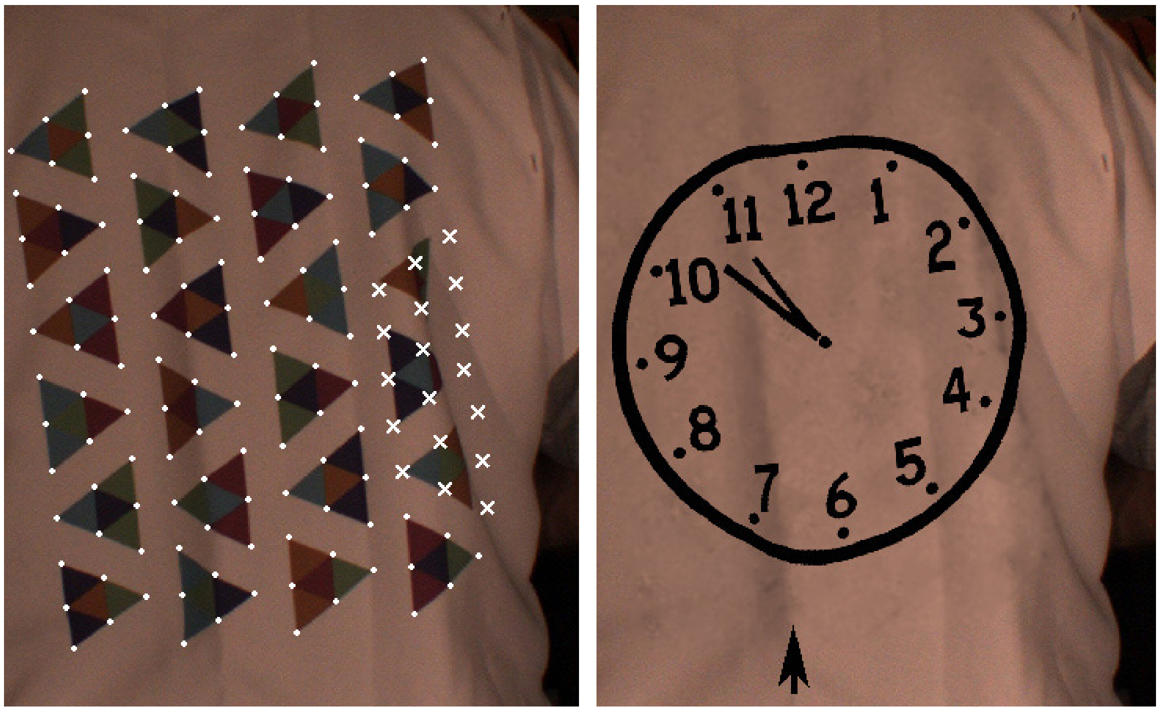


Figure 3.4: In some cases, our estimate of the transformation is poor. On the **left**, vertex locations for missed triangles were approximated inaccurately. Because our method **does not** rely on explicit correspondence to compute an irradiance estimate, the reconstructed image on the **right** does not contain obvious artifacts. While the image appears plausible, the re-textured surface is not physically plausible — the texture and lighting cues disagree. Again, we point out that irradiance captures shape at multiple scales: fine creases (follow the black arrow) and larger folds.

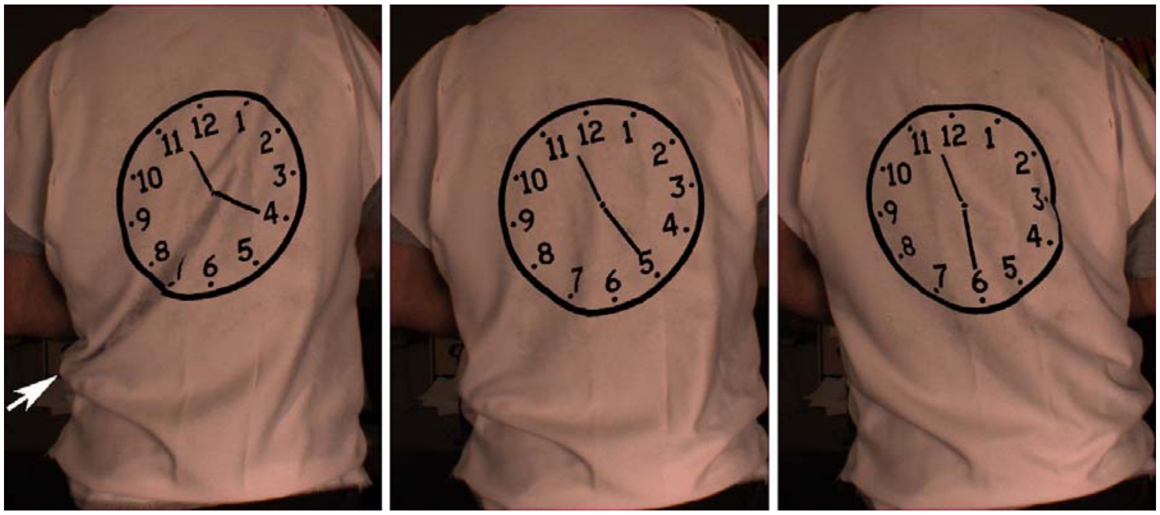


Figure 3.5: Retexturing is not limited to static images — here we retexture with a ticking clock. (there are 4 frames between each image) On the **left**, the white arrow points to a strong folds that pierces the middle of the clock — giving a strong cure about surface shape.

the texture map (derived below), we combine our lighting estimate with the original pixels to get a 'blank' surface. We replace pixels in the textured region, blend nearby pixels, then use the original image pixels for the rest of the image (Figure 3.2).

3.4 Results

We have demonstrated the power of retexturing using irradiance on several videos of deforming non-rigid surfaces, including t-shirts and plastic bags. In general, results using a map are better: large numbers of correspondences provide a better replacement texture (Figures 3.2 and 3.5). However, our irradiance estimation is robust — meaning that irradiance estimates are correct even when the texture map is coarse (Figure 3.7). This is important because irradiance estimates are a powerful cue to surface shape. As a result, denser maps do not provide better estimates of irradiance (Figure 2.4). Different

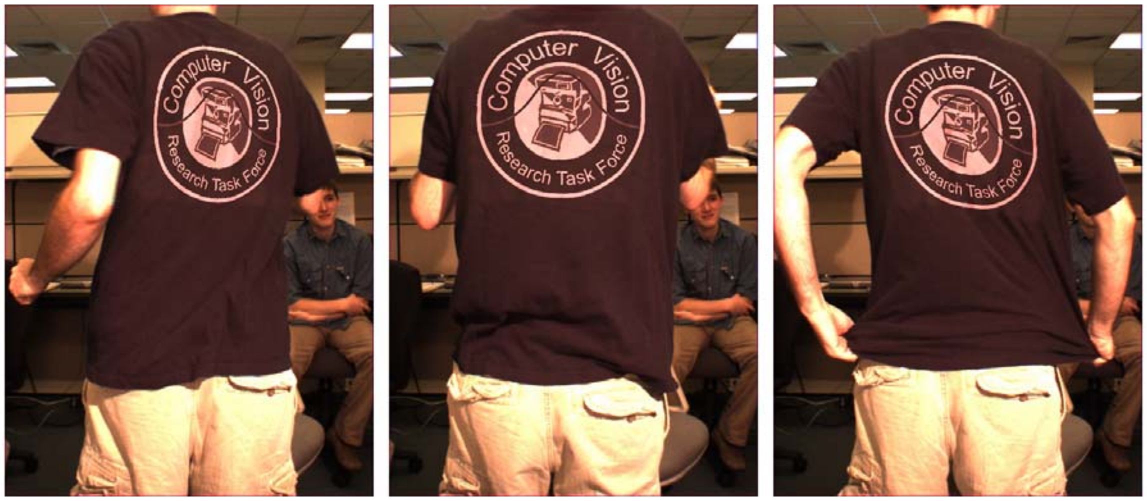


Figure 3.6: Dark albedos present a challenge for our irradiance estimation. The range of intensities is smaller and there is more noise. Our irradiance estimates are smooth and have a distinctly different appearance (look closely at the lower left of the logo). However, the rendered surface is also dark, making errors harder to spot.

background colors do not present a problem: we show results on a shirt with a dark albedo as well (Figure 3.6).

Our results suggest several areas for future work. First, the local method does not interpolate missing triangles well, implying that a hybrid approach may be more effective. Second, our method of interpolating irradiance can be improved: we believe that using texture synthesis could provide more realistic results.

We interpret our results to indicate that surface energy terms may be unnecessary for retexturing. Furthermore, a model that reflects the underlying mechanics poorly can result in significant correspondence errors. A 2D elastic model has difficulty managing the large apparent strains created by folds and occlusions. However, a model that uses the image data itself (without surface energy), such as the model presented in this paper, is enough to retexture.

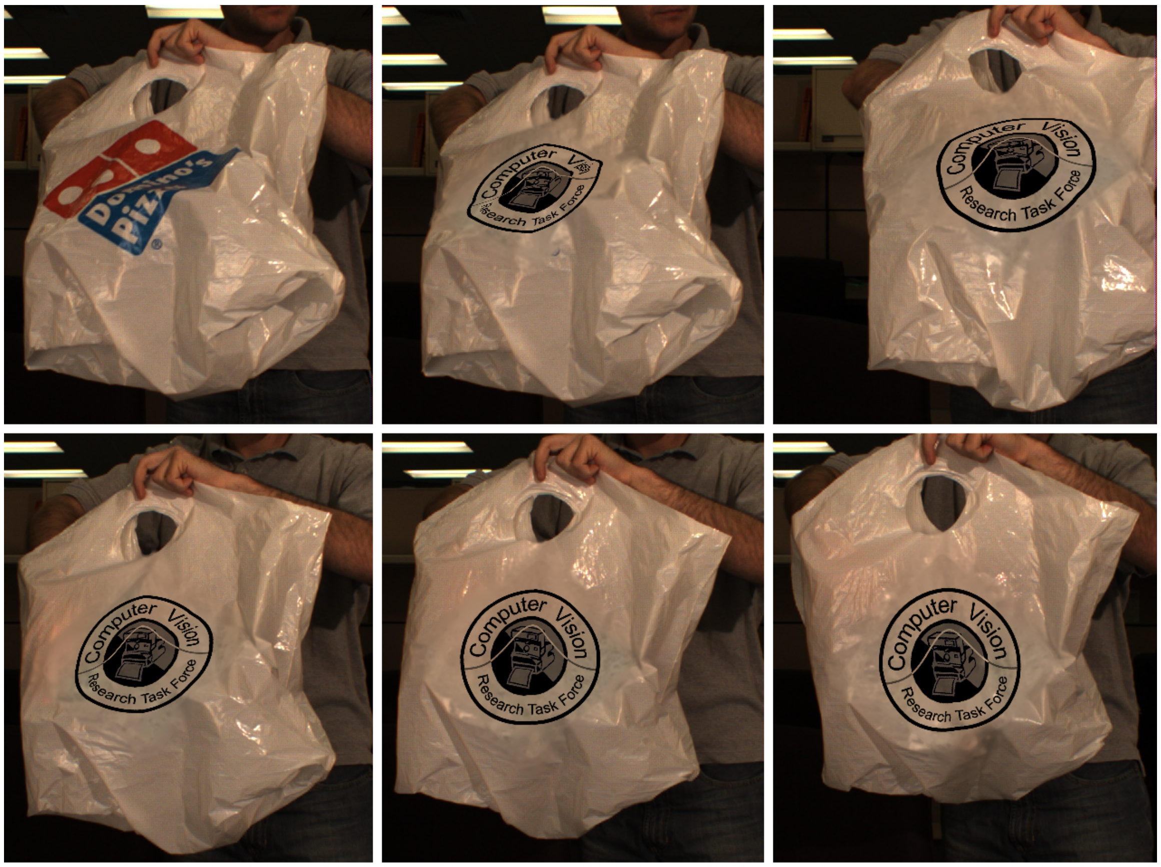


Figure 3.7: Retexturing is not limited to clothing. This plastic bag exhibits a significantly different type of motion from textiles. Since our model does not include a surface deformation term, our detection method continues to work well. In addition, our lighting model can even account for the highlights due to specular reflection.

Chapter 4

Geometry from a Single View

We demonstrate reconstructions from a single view using a combination of shape and texture cues. We show our reconstructions are geometrically accurate by comparison with reconstructions from multiple views. Traditionally, reconstruction techniques are limited by ambiguities in the local cues — and reconstructions are performed by breaking these ambiguities using global consistency. Instead, we break ambiguities locally and only introduce long scale consistency in the finals steps of the geometric reconstruction.

We start with an outline of the reconstruction process. First, we obtain an estimate of the frontal appearance of the texture. This can be supplied manually, or reconstructed automatically from a sequence of images (section 2.4, figure 2.9). Second, we decompose the image into an irradiance map and a texture map (section 4.2.1, figure 4.5). Third, we obtain ambiguous estimates of the surface normals using the frontal texture, the texture map and the assumption of local orthography [Forsyth 01, Forsyth 02]. The normals are disambiguated using a shading model applied to the irradiance map (section 4.2.2). Finally, the surface is reconstructed using perspective effects to break a final concave convex ambiguity (section 4.3).

The resulting reconstructions are highly accurate – estimating geometry within four percent. Only a small number of papers numerically quantify the accuracy of a geometric reconstruction from a single image — and none of them in similar situations using real data. Forsyth reconstructs the shape of a synthetic image of a textured sphere [Forsyth 02] and Savarese et al. reconstruct the shape of a cylindrical mirror using the reflection of a calibration pattern [Savarese et al 04].

4.1 Background

Shape-from-Shading Starting in [Horn 70], most work in shape-from-shading makes simplifying assumptions: the scene is illuminated with a point light source, the surface of the object is lambertian and inter-reflections are minimal. Methods typically use either local intensity or local gradient information. Constraints such as surface smoothness, integrability and boundary conditions break ambiguities. However, scenes exhibit effects from mutual illumination [Forsyth and Zisserman 91], surfaces are not entirely lambertian and often there are multiple light sources. Even in the cases where the assumptions hold, current reconstruction methods are unreliable [Zhang et al 99].

Shape-from-Texture Texture normals can be computed as a transformation between an observed patch and a frontal view by assuming that the patch is locally flat. Local estimation of surface normals is limited by two problems: one must know the frontal appearance of the textured patch, and each estimated normal is actually an ambiguous pair:

the estimated normal and its mirror across the viewing direction. Forsyth [Forsyth 01, Forsyth 02] and Lobay and Forsyth [Lobay and Forsyth 04, Lobay and Forsyth 06] focus on estimating the frontal texture from a single view of a repeating pattern and use surface smoothness to resolve the texture normal ambiguity. Loh and Hartley also use perspective effects to break ambiguities [Loh and Hartley 05].

We take a different stance on both topics. First, we estimate the frontal appearance of a non-repeating texture pattern using multiple images of the deforming object. Second, we break ambiguities in texture normals using shading cues. There is no reason to limit combinations of the two techniques. One could estimate the frontal pattern of a repeating texture and use shading cues to break ambiguities. Though less reliable, one could also estimate the frontal view of an un-shaded deforming surface in multiple views and use smoothness to break texture normal ambiguities.

Combined Shading and Texture Choe et al [Choe and Kashyap 91] reconstruct the 3D shape of shaded images of physically generated random textures based on surface irregularities. They focus on models of natural textures (reconstructing the shape of a tree trunk) and do not consider using shading to break texture ambiguities.

Deformable Surface Tracking Several methods have been proposed to track nonrigid motion [Tsap et al 00, Pilet et al 05]. Pilet et al [Pilet et al 05] describe a method to detect the surface deformation using wide-baseline matches between a frontal view and the image to estimate a transformation smoothed using surface energy. This method has difficulty stabilizing texture for 3 reasons: there are few reliable keypoints in textures we consider; because internal keypoints are used, boundaries are not tracked well; and the surface energy term makes it difficult to track highly deformable objects. Also, they cannot obtain an irradiance estimate using the surface track because errors in correspondence would make it impossible.

4.2 Estimating Normals

Our method requires the extraction of accurate normals to estimate geometry. To compute surface normals, we estimate ambiguous normals using texture cues, then break the ambiguity using a lighting model and shading cues. As part of the process, we decompose images into two components: albedo and irradiance — which correspond to the two aspects of normal estimation: texture and shading.

Following previous work in shape-from-texture and shape-from-shading, we emphasize the reliable nature of texture normals. While both shading and texture provide (ambiguous) cues about the surface normal, we consider texture to be a stronger cue: the ambiguity is two-fold as opposed to a 1D ambiguous family and the assumptions that underlie texture normal estimation are more reasonable in practice (local planarity vs lambertian scenes without inter-reflection). Later in this thesis, we show that texture normals are often correct within a few degrees when compared across different views of the same object (section 7.3).

Because there are several distinct ambiguities, we adopt the following terminology: **Texture normal ambiguity** (or **texture ambiguity**) refers the two-fold ambiguity in estimating a single normal from an observed texture. **Combined cue ambiguity** refers to the uncommon texture ambiguity that remain ambiguous after using shading cues to break ambiguities. Finally, **scene ambiguity** refers to the concave convex ambiguity that results from confusion about the source of the lighting.

In general, we break **texture normal ambiguity** using shading information. We use belief propagation and surface smoothness to break the infrequent **combined cue ambiguity**. Finally, we use perspective effects on the scale of the scene to eliminate **scene ambiguity**.

4.2.1 The Lighting Model

In this section, we describe the lighting model and use it to decompose the image into two distinct parts: a texture map and an irradiance map. We detail the process of breaking the texture ambiguity in the next section and subsequently discuss the determination of the lighting parameters in section 4.2.3.

Following previous work in shape-from-shading, we assume that our scene is lit by a combination of a point light source and a non-directional constant background illumination. Using $\mathbf{L}$ as the direction to the light source, $\mathbf{N}$ as the surface normal, ρ_d as the surface albedo, L_p as the point source intensity and L_a as the background illumination intensity, the intensity $\mathcal{I}_i$ of a pixel i in the image is:

$$\mathcal{I}_i = \mathbf{L} \cdot \mathbf{N} \cdot \rho_d \cdot L_p + \rho_d \cdot L_a$$

If we know the surface albedo ρ_d and the lighting model parameters ($\mathbf{L}$, L_p, L_a), then we can compute the angle between the light and the normal (although the normal direction itself is limited to a 1 parameter family) and subsequently determine an irradiance image — one where the surface albedo is constant for the entire image.

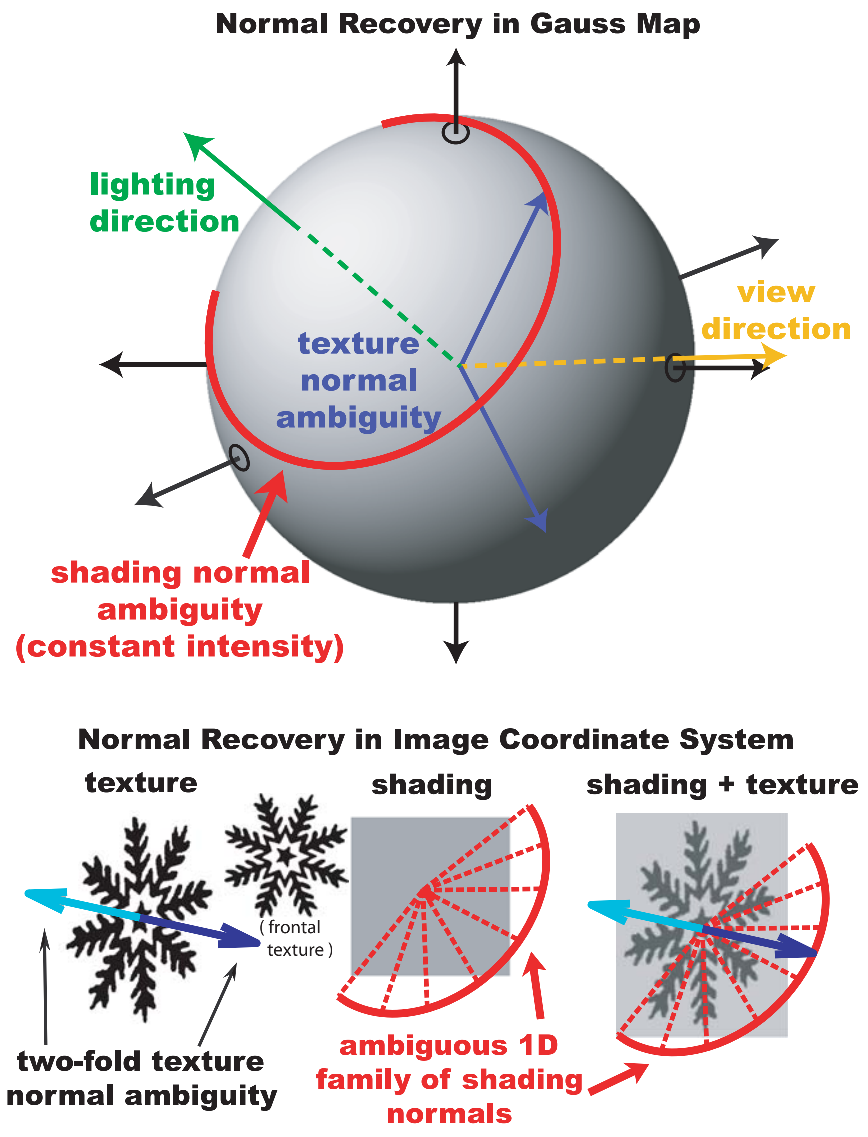

Normal Recovery in Gauss Map
lighting direction
view direction
texture normal ambiguity
shading normal ambiguity (constant intensity)
Normal Recovery in Image Coordinate System
texture
shading
shading + texture
(frontal texture)
two-fold texture normal ambiguity
ambiguous 1D family of shading normals

Figure 4.1: While local cues for estimating normals are ambiguous when considering shading or texture alone, the ambiguities can be broken in most cases by combining cues. On the **top**, we view a gauss map of a sphere shaded with a point light source. The red arc denotes an isophote (constant pixel intensity) — a single measurement of image intensity constrains the normal to this curve. Observations of texture produce a two-fold ambiguity symmetric across the viewing direction. In most cases, the ambiguities do not line up (for exceptions, see figures 4.2 and 4.3) and the normal can be determined. On the **bottom**, we show the same phenomenon from the viewpoint of the camera. An observation of a textured patch can have two possible normals, while a single measurement of intensity has a 1 parameter family of normals. By combing cues, the normal can be determined.

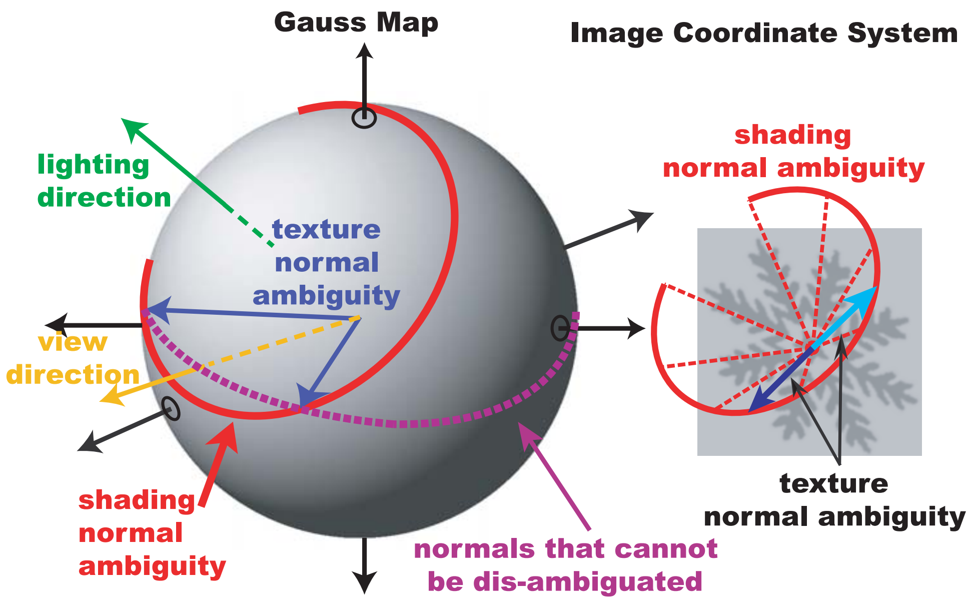


Figure 4.2: Shading usually breaks texture normal ambiguities; however, there are cases when the two ambiguities line up and the ambiguity can not be broken. In this image, the two ambiguous texture normals lie on the same arc of constant irradiance, meaning that the reconstruction is still ambiguous. The purple line shows the set of all normals where shading will not break the texture ambiguity: we call this **combined cue ambiguity**.

Estimating Irradiance: Following the work in [White and Forsyth 06b], we focus on screen print items (ones that have a finite set of colors) and use a classifier to determine which dye color generated each pixel. The output of the classifier is the texture map for the image. To build the irradiance map, we estimate an image with constant albedo ($\rho_d = 1$). This image can be generated by dividing the observed pixel intensity $\mathcal{I}_i$ by the albedo for the dye color (ρ_d). While a texture map may provide a rough guide to the surface albedo, the color classifier is much better because it is not effected by small alignment errors.

4.2.2 Breaking Texture Ambiguity with Shading

Figure 4.1 provides a geometric perspective on the process of breaking texture ambiguity. Each observation of irradiance ($\mathcal{I}_i$) confines the surface normal to a 1 parameter family of possible normals. Each textured patch limits the normal to a set of size two. In the ideal case, the intersection of these sets produces one and only one normal. However, in practice the sets do not intersect. Because we believe normal estimates from texture to be substantially better than normal estimates from lighting, we use the shading cue to break the texture ambiguity.

We formulate the problem of breaking the texture ambiguity as an error between the irradiance computed using texture normals under the lighting model and the irradiance image. Writing N_{i1} and N_{i2} for the two possible texture normals and δ_i as an indicator variable to switch between the two choices, we write the cost as:

$$\sum_i (\mathbf{L} \cdot N_{i1} \cdot L_p + L_a - \mathcal{I}_i)^2 \delta_i + (\mathbf{L} \cdot N_{i2} \cdot L_p + L_a - \mathcal{I}_i)^2 (1 - \delta_i)$$

If we assume that L_a, L_p and $\mathbf{L}$ are known, then the process of breaking the texture ambiguity (or, correspondingly the value of δ_i) is simple: for each ambiguous normal choose the normal with the lower cost.

However, when the lighting (cost) for the two possible texture normals is the same, shading will not break the texture ambiguity, resulting in a **combined cue ambiguity**. While this formal ambiguity is limited to a single arc on the gauss map, because of unmodeled effects the region of remaining ambiguity is a thick line region around the formal ambiguity. Even worse, when the lighting direction and the viewing direction are the same, shading cues will not break any texture normal ambiguities (figure 4.3).

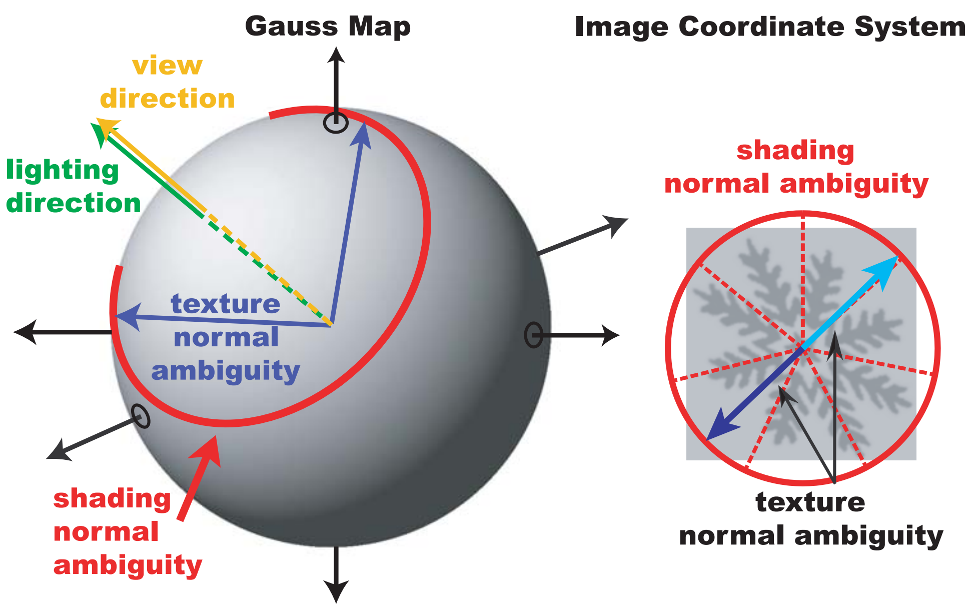


Figure 4.3: When the lighting direction and the viewing direction are the same, none of the normals can be disambiguated.

4.2.3 Determining the Lighting Direction

Lighting parameter recovery is based on two observations: First, texture cues provide information about the accuracy of lighting parameters. Second, because the lighting model has a small number of degrees of freedom (four: two for the lighting direction **L**, one each for the point light intensity L_p and the background intensity L_a) it is not necessary to search over the 2^D values of δ_i to find an optimal set of parameters. (D is the number of normals)

In fact, for a given lighting model there are not 2^D values of δ_i because some choices necessarily have higher costs than others. There are in fact $O(D^4)$ by the following argument. The value of δ_i changes at points where either of the two linear constraints is true:

$$(\mathbf{L} \cdot N_{i1} \cdot L_p + L_a - I_i) = (\mathbf{L} \cdot N_{i2} \cdot L_p + L_a - I_i)$$

$$(\mathbf{L} \cdot N_{i1} \cdot L_p + L_a - I_i) = -(\mathbf{L} \cdot N_{i2} \cdot L_p + L_a - I_i)$$

This means that the δ_i are constant within the cells of the arrangement of hyperplanes given by these constraints, and there are $O(D^4)$ such cells. Given a particular δ_i, the error is a convex function within a convex cell and can be minimized by standard methods. Instead of building the arrangement, we grid the variables (also $O(D^4)$) and start an optimization problem at the minimal grid point.

We score a choice of lighting parameters using the residual under the lighting model, assuming that each normal is charged for the minimum penalty:

$$Q(\mathbf{L}, L_p, L_a) = \sum_{\text{normals}} \min(\text{error}_1^2, \text{error}_2^2)$$

We cannot directly apply gradient descent to optimize this function since the cost function is not differentiable. In the implementation, we search in two steps. First, we perform a coarse scale grid search by iterating over a list of values for each parameter. Using this rough solution to fix the disambiguation of the normals, we perform gradient descent. While in theory the light source could move enough to force a change in normal disambiguation, in practice the light source doesn't move much. More importantly, as long as the source doesn't move much, the two costs are comparable in value.

The location of the lighting source, however, is still ambiguous up to a **scene ambiguity**: a mirror light source across the viewing direction will produce the same error. The two light sources translate into an ambiguity in the sign of the depth for the entire image (or a concave convex ambiguity). Even in scenes with little depth, this ambiguity can be broken with perspective effects. (section 4.3).

4.2.4 Breaking the Remaining Ambiguities

As noted in previous sections, there are some normals that are intrinsically ambiguous using shading and texture cues. We break the ambiguity in these normals by assuming that the surface is smooth and that the surface normals change slowly – therefore neighboring normals should be similar. While previous approaches used this assumption to break all ambiguities, our method is more reliable because only a small set of normals have combined cue ambiguity.

We detect ambiguous normals by comparing the lighting errors between the two texture normals. Small differences indicate ambiguity. We threshold, asserting that roughly 90% of the gauss map is not ambiguous. The confidence for remaining 10% of normals is weighted linearly by the difference in errors between the two texture normals. We set up a new graph, where the nodes in the graph are the triangles in the surface mesh and the edges in the graph are edges in the mesh. Using a probability model that favors neighboring normals that agree, a few steps of loopy belief propagation on this graph produces unambiguous results.

4.3 Recovering Geometry

To recover a full 3D model from disambiguated normals, we iterate over two stages: (1) compute scene depth assuming orthography using normals; (2) adjust the point locations

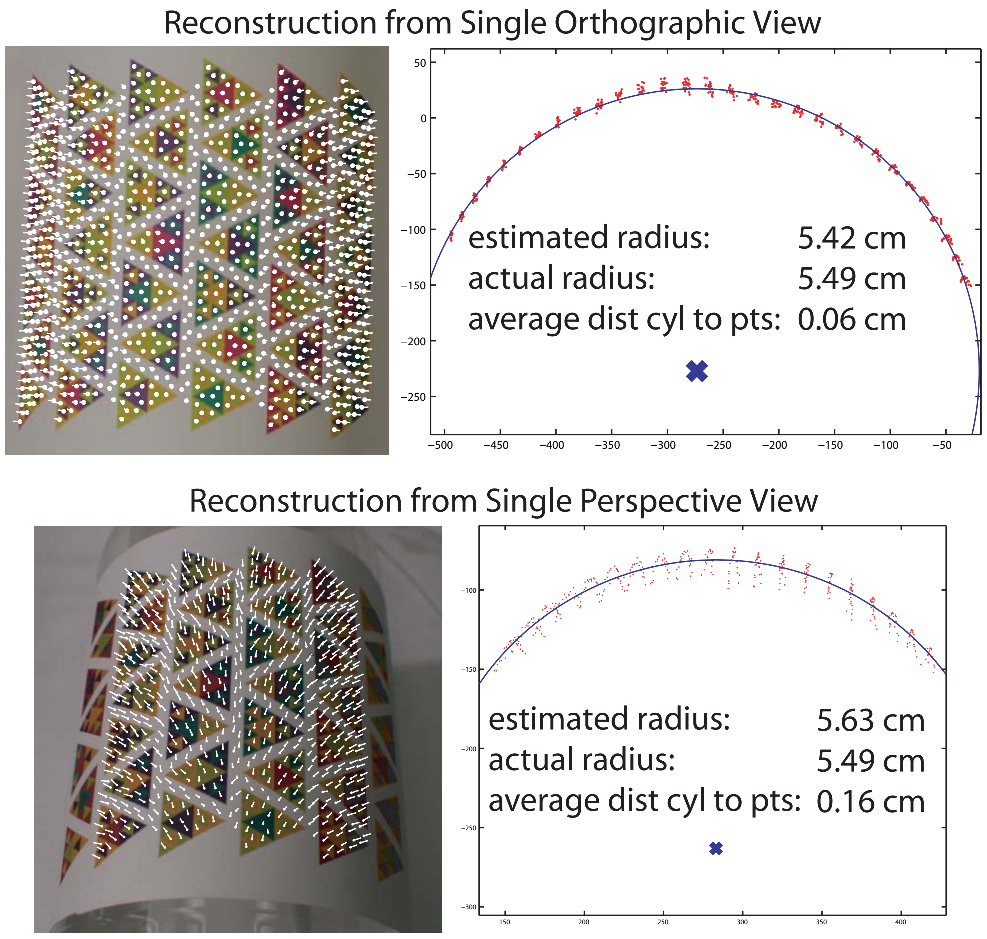


Figure 4.4: Using both texture and lighting cues, accurate models of shape can be made **from a single view**. Using a custom pattern taped to a cylinder (a 2L bottle), we reconstruct the shape twice: in an image with minimal perspective effects (**left**) and in an image with more prominent perspective effects (**right**). In both cases, the reconstructions are performed using the same code, assuming orthography on the scale of a triangle, and perspective effects on the scale of the scene. The 3D reconstructions are viewed down the axis of the best fit cylinder — where the blue x denotes the estimated axis and the arc denotes the best fit surface. The red point are the vertices of the points. The quality of the fit is the average distance between the cylinder and the point cloud — in the orthographic case 2.98 pixels (0.637 mm) and 4.97 pixels (1.57 mm) in the perspective case. Errors in estimating the radius: 2.82 pixels (0.602 mm) in the orthographic image and 4.46 pixels (1.41 mm) in the perspective image.

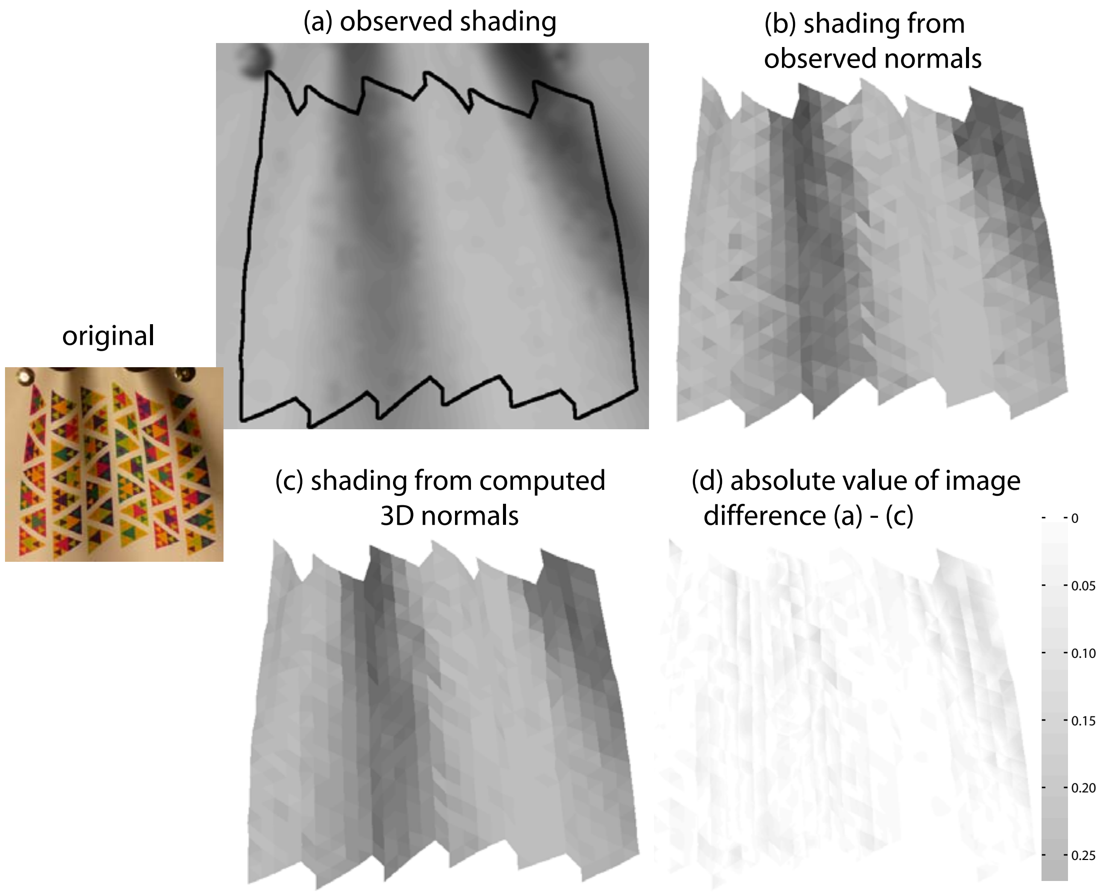

(a) observed shading
(b) shading from observed normals
original
(c) shading from computed 3D normals
(d) absolute value of image difference (a) - (c)
0
0.05
0.10
0.15
0.20
0.25

Figure 4.5: By comparing the original image shading with two different shading reconstructions, we show (1) that agreement between shading and texture cues is fairly robust and (2) that estimating 3D geometry effectively smoothes over noisy normals. In (a), we show an image of the shading (derived from the frontal image in the inset, also shown in figure 4.7). This shading was estimated using the method in [White and Forsyth 06b]. In (b), we render the lighting assuming our simplified lighting model, using the disambiguated normal to generate appropriate shading. Because the normals are calculated up to two-fold ambiguity from texture cues, we can interpret the agreement between this image and the extracted shading image as the agreement between texture and shading cues. Finally, in (c), we render the normals from the 3D geometry in the same way. While no additional information about shading was incorporated between the middle and right images, by estimating 3D geometry, we effectively smooth the normal field — producing more realistic results. The difference image (d) shows that the resulting errors are small.

to account for perspective effects. Because of **scene ambiguity**, the sign of the depth is ambiguous and we start the process twice: once for each sign of the depth. By picking the solution with minimal error, we get a metric reconstruction of the scene from a single view.

We assume that perspective effects are not observable on the scale of an individual triangle but are only observable (albeit minimal) on the scale of the scene. If perspective effects were viewable on the scale of the element, then either (a) the element is large and not particularly descriptive; or (b) the element is at a highly oblique angle and detection is unreliable. In practice, the validity of these assumptions is demonstrated by two empirical facts: (1) texture normals computed assuming local orthography are accurate; (2) in practice, scene ambiguity can be resolved in images with weak perspective effects (figure 4.4).

4.3.1 Orthographic Depth from Normals

We estimate the orthographic depth from normals using a mesh constructed out of triangles in the image plane. Using gradient descent, we constrain the x and y locations of the vertices while allowing the z value to vary. Our objective function is a combination of two costs: agreement between the estimated and 3D normal and strain in the 3D mesh. Instead of computing the alignment between the 3D normal (n_i) and the image normal, we use tangents to the image normal (t_i^1, t_i^2) to make our cost function quadratic:

$$\mathrm{c}_{\text{normals}} = \sum_{i \in \text{normals}} (t_i^1 \cdot n_i)^2 + (t_i^2 \cdot n_i)^2$$

However, this problem has local minima: normals that flip across a tangent are unlikely to return — typically causing a single vertex to appear far from the actual surface.

We include the strain term to regularize the solution. Using L as the rest length of a triangle edge and ΔL as the observed change in length, strain is $\frac{\Delta L}{L}$. By penalizing strain, the mesh acts like a set of springs. Strain penalties have been useful in graphics [Provot 95] and cloth surface reconstruction [White and Forsyth 05]. We include strain in the optimization by adding a cost the square of the strain to our objective function.

This method of estimating the depth smoothes the normal field because it computes the minimum error solution to an over constrained problem. Roughly speaking, our cost has two constraints per normal (one normal per triangle) with one degree of freedom per vertex. Although mesh connectivity varies, we typically have more triangles than points — often by nearly a factor of two. As a result, there are roughly four times as many constraints as free variables. Using the lighting model, the estimated 3D normals produce smaller errors in estimated irradiance than the 2D normals computed from the surface texture (figure 4.5).

4.3.2 Adjusting for Perspective Effects

We incorporate perspective effects by computing the image locations for points viewed using a hypothetical orthographic camera. Using these modified points, the depth estimation procedure in the previous section can be used on images with perspective effects. To compute the hypothetical orthographic image, we estimate the depth between the scene and the camera and use this depth to adjust the observed (x,y) image locations to their orthographic equivalent. The distance between each point and the camera center is multiplied by a depth adjustment term based on the relative distance between the point and the camera. We assume that the camera center is the middle of the image and define z_i as the

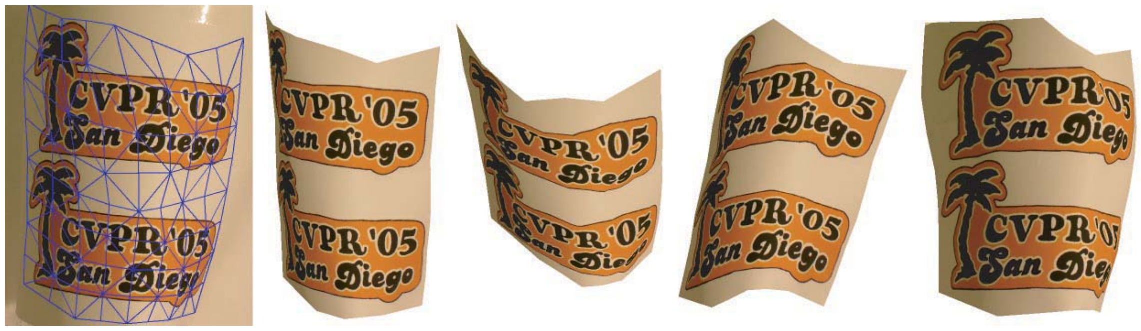


Figure 4.6: Using the frontal appearance estimated in figure 2.9, we use shading and texture cues to estimate the surface normals and consequently the 3D geometry. The original image is shown on the **left**. The four images on the **right** are re-renderings of the extracted geometry shown from new viewpoints. While we do not have ground truth for this 3D reconstruction, we believe the geometry to be relatively accurate because (a) our method is accurate in similar settings (figures 4.4, 4.8 and 4.7), and (b) the re-rendered images agree with our aesthetic sense of the shape.

estimated depth of point i and $\bar{z}$ as the average depth of the scene:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{d + z_i}{d + \bar{z}} \begin{bmatrix} x \\ y \end{bmatrix}$$

Using this adjustment term, nearby points move closer to the center of the image, while farther points move farther to eliminate perspective effects. We optimize by running gradient descent over the depth value d, using the same cost function as in the previous section.

4.4 Feature Representation

We present results on two separate types of data — ones where we print a custom, highly localizable pattern and data where the pattern is naturally occurring. Methods for detecting and tracking these texture were covered in chapter 2. In both cases, our tracker outputs triangles that represent a texture deformation between a frontal view of the surface and the observed view. This mesh structure implicitly gives us ambiguous texture normals — for convergence reasons, we use unit quaternions to parameterize the normal.

4.5 Experiments

4.5.1 Implementation: Normals from Texture

Extracting ambiguous surface normals from texture is fairly commonplace, and has proved to be exceedingly accurate. In our implementation, we make a minor modification to the standard technique by using unit-quaternions to parameterize the transformation instead of the more familiar composition of rotation, tilt, rotation. We represent the transformation between the frontal view of the texture and the observed view as the coordinates of the triangle in each of the two views, constructing the relationship: $p_{\text{target}} = A \cdot p_{\text{source}}$. We can think of A as a composition of a scale (since we don't know the scale of the triangle) and the upper 2x2 section of a 3D rotation matrix. The points (p_{target}, p_{source}) exist in the 2D world of the image plane. In [Forsyth 02], Forsyth effectively parameterizes this transformation in terms of three Givens rotations (rotation, tilt, rotation). We have found, in practice, that non-linear minimization using this parameterization is subject to gimbal lock, especially for estimating normals for near frontal textons. Instead, we parameterize the rotation using unit-quaternions. We randomly initialize and perform gradient descent. Rarely, the system fails to converge and we must choose a new random starting point.

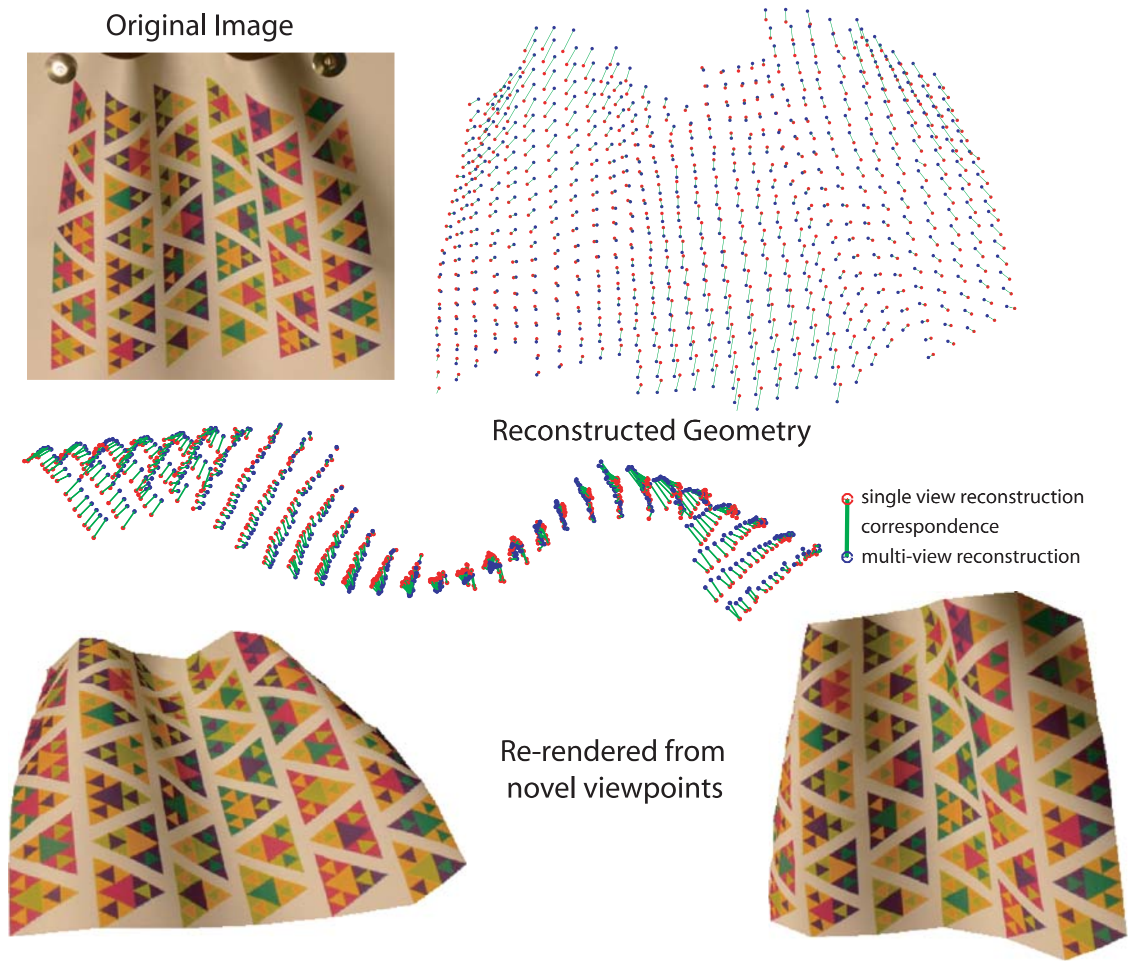

Original Image
Reconstructed Geometry
single view reconstruction
correspondence
multi-view reconstruction
Re-rendered from novel viewpoints

Figure 4.7: We demonstrate the quality of the shape reconstruction by recovering the geometry of an unknown object, and compare to a more accurate reconstruction obtained from multi-view geometry. In the reconstructed geometry, the red points are the reconstruction obtained from single view texture and shading while the blue points are obtained using multi-view reconstruction from 4 images. The green lines show the correspondence between the two methods — longer green lines indicate larger discrepancies. Because of the accuracy of multi-view reconstruction (reprojection error of 0.37 pixels and 0.0082 cm or 82 μm), we can reasonably consider this discrepancy to be the error in the single view reconstruction. The average discrepancy is 10.6 pixels or 2.335 mm on an object with a side-length of 7.8 cm. Note that this test is more challenging than the tests in figures 4.4 and 4.8 because it also penalizes texture slip — yet results are still accurate to roughly 1 in 40. We use the multi-view calibration method described in [White and Forsyth 05].

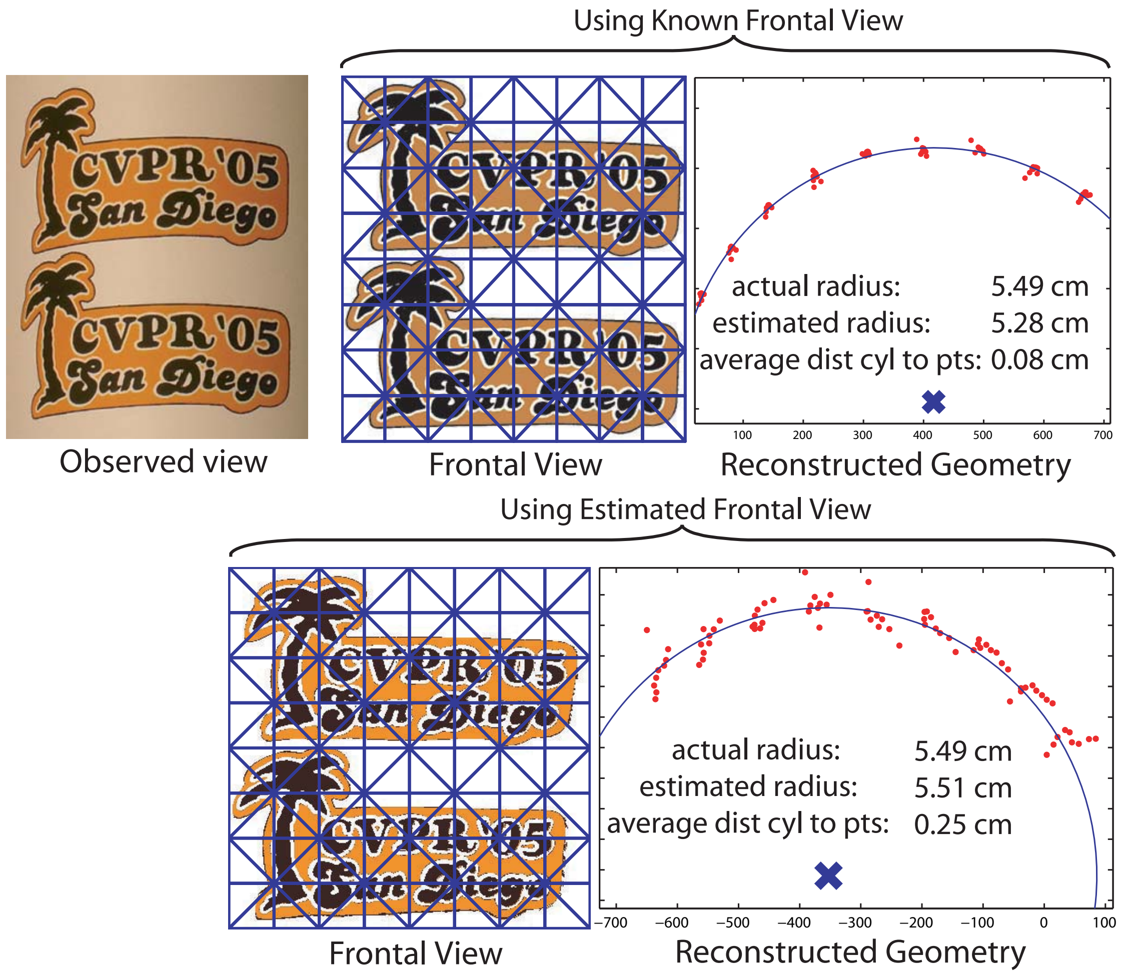


Figure 4.8: Again, we compute the shape **from a single view** using texture and shading cues. Instead of requiring a custom pattern to compute the shape (figure 4.4), we use a supplied frontal view of the texture on the **right** and our estimated frontal view from figure 2.9 on the **left**. Using the supplied frontal texture the 3D point cloud is more consistent with the cylindrical model than the points generated using the estimated texture — however both textures produce accurate estimates of the radius of the cylinder. We use a deformable model to fit the mesh to a novel image of the pattern taped to a 2L bottle (section 4.4). This transformation allows us to compute ambiguous normals. A subsequent stage determines the location of the light source (section 4.2.3), breaks the ambiguity (section 4.2.2) and uses the normals to produce a 3D mesh (section 4.3). In images of the reconstructed geometry, we view the 3D point cloud down the axis of the cylinder. Here, the estimate of the radius is within 16.85 pixels or 2.12 mm. The average distance between the cylinder and the point cloud is 6.18 pixels or 0.78 mm.

4.5.2 Results

We empirically establish the validity of our method by a sequence of experiments of increasing difficulty.

First, we focus on images of custom textures with known frontal appearance. These experiments are done in 3 forms: one with an orthographic image of an object of known geometry (figure 4.4), one with a perspective image of an object known geometry (figure 4.4) and finally a perspective image of a surface of unknown geometry (figure 4.7). In the first two cases, we reconstruct without knowledge of the geometry, then compare the results to our model. In the third case, we use multi-view geometry to reconstruct the shape and compare the results between the methods. In all cases, we generate highly accurate 3D models – errors in 3D geometry are less than three percent.

Second, we consider arbitrary textures. We reconstruct the shape of a known geometry under two circumstances: a user-provided frontal texture, and an estimated frontal texture (figure 4.8) generated from eight views of the different deformed surfaces (figure 2.9). Despite the more challenging arbitrary texture, we are still able to estimate geometric parameters within four percent. The estimation of the frontal texture causes only minor increases in fitting error compared to a user supplied frontal texture.

Finally, we reconstruct the frontal appearance of an arbitrary texture from several deformed views (figure 2.9) and use this frontal texture to estimate the geometry in one of the observed views (figure 4.6). We test this method visually on a video sequence of a deforming t-shirt by reconstructing the geometry in one view and rendering the result from a alternate view and compare with an actual recording (figure 4.9).

4.6 Discussion

We have demonstrated high quality reconstructions from single views. Our method requires a texture estimate, which could be obtained as prior knowledge, by observing a moving object, or possibly from repeated texture. We think that this method of obtaining shape has several applications: including object recognition and scene understanding, graphics, and physical measurement. The quality of our reconstruction provides a middle ground between the flexibility of single cue single view reconstruction and the accuracy of multi-view techniques.

We have considered the estimation of normals to be a process that simply breaks texture ambiguities. However, figure 4.5 suggests that we could take the process a step further — and produce a normal estimate that minimizes the error between the texture estimate and the shading estimate.

We have put considerable effort into reconstructing shape *without* boundary conditions. However, boundaries provide significant shape cues. A method that identifies boundaries and provides additional constraints would add another information source and probably provide more accurate reconstructions.

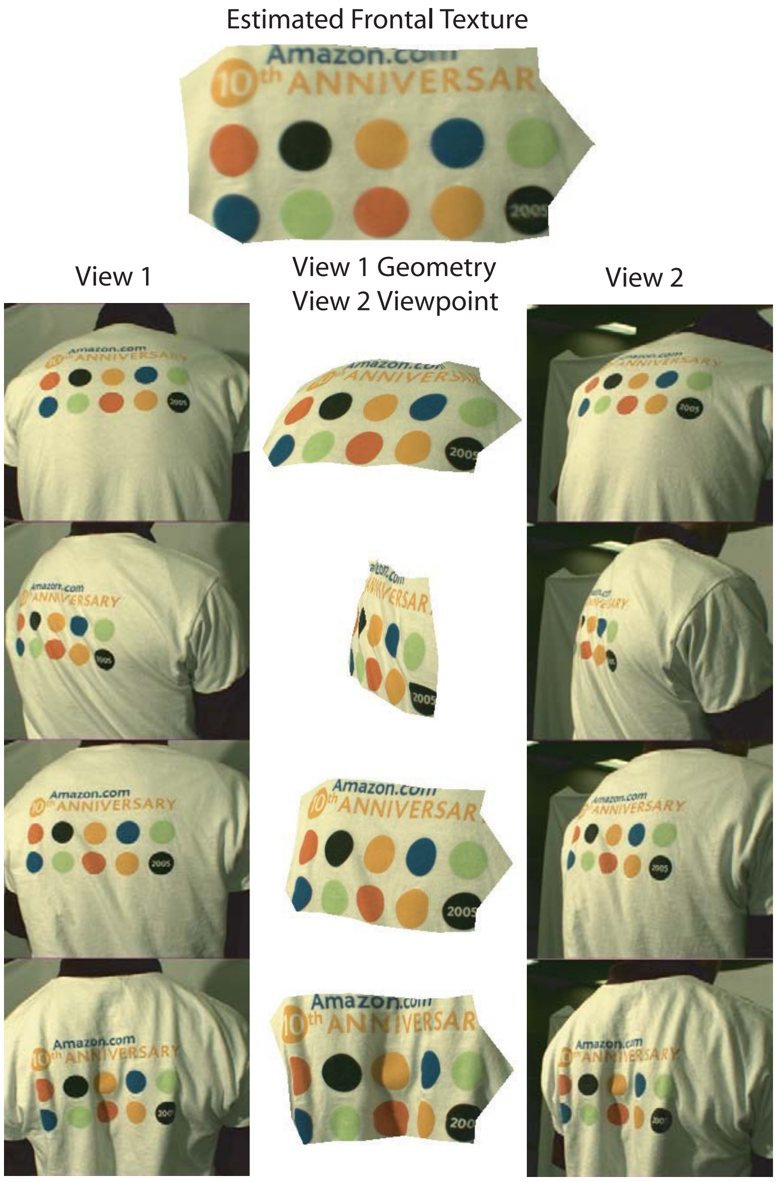


Figure 4.9: We use a dynamic sequence of the back of a t-shirt to recover the frontal view (**top**) and subsequently recover surface geometry (**bottom**). In this case, we film the scene from two angles: one that we use to compute the geometry and the frontal view and another to use for comparison. By rendering geometry from the first view from the perspective of the second view, we can visually verify the quality of the result. The shape is only recovered for the localizable textured region of the cloth as described in figure 2.7. The frontal texture was computed from a total of 24 frames.

Part II

Multiview

Chapter 5

Capturing Cloth in Multiple Views

In this part, we capture the 3D shape of moving cloth using multiple synchronized video cameras. The output of this process is a sequence of triangle meshes with static connectivity and with detail at the scale of individual markers in both smooth and folded regions. In chapter 6, we focus on the problem of correspondence: between views, how do we estimate which points correspond? In the chapter 7, we perform the subsequent computation: calibration and reconstruction.

This setup is more costly than previous setups and is suited to a studio environment for creating and editing video games and film. We compute locations of points in space using marker correspondence across the cameras. Correspondence is determined from color information in small neighborhoods and refined using a novel strain pruning process. Final correspondence does not require neighborhood information.

The remaining portions of this thesis are all devoted to the same problem: building geometric models of moving cloth from multiple views. This chapter is focused on the

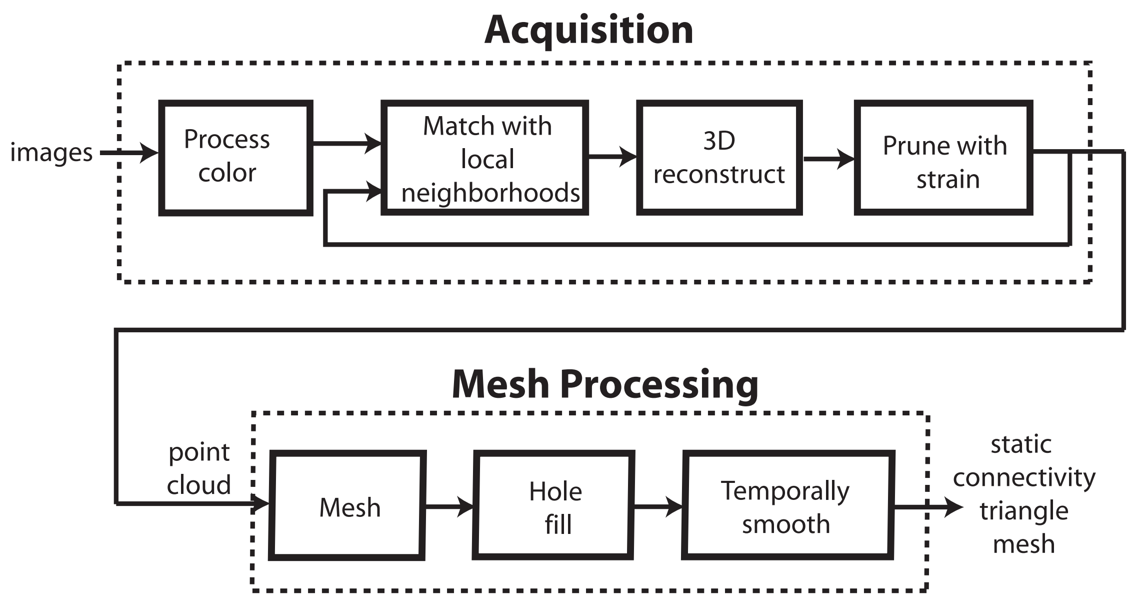


Figure 5.1: We make animated meshes for graphics applications by producing a sequence of parameterized surface meshes with same connectivity in every frame. We capture cloth into two stages: acquisition and mesh processing. In acquisition, we take raw images and convert them into a 3D point cloud. In mesh processing, we triangulate the mesh, fill the holes and temporally smooth.

least studied part of the problem: establishing an accurate correspondence between views. Chapter 7 focuses on the geometry of multiple views: including issues in calibration, 3D reconstruction and integration of volume cues into reconstruction. Finally, chapter 8 looks at a data driven hole-filling technique to fill occluded regions. This technique is then extended to data driven animation.

5.1 Overview

We capture the motion of cloth using multiple video cameras and specially tailored garments. The resulting surface meshes have an isometric parameterization and maintain static connectivity over time. Over the course of roughly half a dozen papers on cloth capture a prevailing strategy has emerged. First, a pattern is printed on the cloth surface such that small regions of the pattern are unique. Next, correspondence is determined by matching regions across multiple views. The 3D location of a region is determined by intersecting rays through the corresponding observations in the image set (figure 6.1). Reconstruction is done independently on a frame by frame basis and the resulting data is smoothed and interpolated. Previous work, such as [Scholz et al 05], yields pleasing results.

Little work has been done to capture garments with folds and scenes with occlusion. We use **folding** to refer to local phenomena such as the wrinkles around a knee and **occlusion** to refer to large scale effects such as one limb blocking the view of another. Folds and occlusion are common, especially when dealing with real garments such as pants where limbs block interior views and cloth collects around joints. Both phenomena are symptoms of the same problem: views of the surface are blocked by other parts of the surface. However, there is a distinction in scale and different methods are required to solve each problem.

When a surface is heavily folded, contiguous visible regions are often small and oddly shaped. In these regions correspondence is essential for detailed reconstruction yet can be challenging to identify. We solve the correspondence problem both by improving the pattern printed on the surface of the cloth and by improving the method used to match regions. Our method gets more information per pixel than previous methods by drawing from the full colorspace instead of a small finite set of colors in the printed pattern. Additionally, because cloth cannot stretch much before ripping, we use strain constraints to eliminate candidates in an iterative search for correspondence. In combination, these two modifications eliminate the need for neighborhood information in the final iteration of our algorithm. As a result, we determine correspondence using regions that are 25 times smaller than previous approaches (figure 6.3).

On the other hand, many regions on the surface are impossible to observe due to occlusion. We fill these holes using reconstructions of the same surface region taken from other points in time. We found that MeshIK ([Sumner et al 05]), a tool originally developed for mesh editing and animation, is appropriate for filling holes in cloth. In fact, we show that MeshIK is well-suited to cloth data in particular and we use it to bind reconstruction of our pants to motion capture data.

We suggest two tools to evaluate marker-based capture systems. The first, markers per megapixel, is a measure of efficiency in capture systems. Efficiency is important because camera resolution and bandwidth are expensive: the goal is to get more performance from the same level of equipment. This metric is designed to predict scaling as technology moves from the research lab to the professional studio. The second tool is information theory: we look at the predictive power of different cues in a capture system. By doing simple bit calculations, we direct our design efforts more appropriately.

5.2 Previous Work

Previous work in cloth motion capture has focused on placing high density markers in correspondence between multiple views. The primary challenge is to increase marker density while correctly assigning correspondence between markers. We suggest **markers per megapixel** as an appropriate metric for comparison (figure 5.2) because it measures the

Work	Megapixels	Markers†	**Markers per Megapixel**
Park 2006	48	≤ 350	$\leq$ **7.3**
Tanie 2005	10	407.9	**40**
Guskov 2003	0.9	≤ 136	$\leq$ **148**
Scholz 2005	8	≤ 3500	$\leq$ **434**
Sleeve	15	7557	**504**
Pants	2.5	2405.3	**979**

Figure 5.2: We suggest **markers per megapixel** as a comparison metric. Because pixels are expensive, efficient use of pixels is necessary. In the pants video, our markers average 56 pixels per camera: the rest of the pixels are consumed by multiple views and background (discussed in section 6.4.2). †When possible we compare *recovered* markers, however some papers exclusively report total markers.

method instead of the equipment. Most high density full frame-rate capture has focused on cloth, however, there has been some recent work enhancing human motion capture [Anguelov et al 05, Park and Hodgins 06]. Because these methods require affixing individual markers, they have far fewer markers per megapixel.

When working with cloth, markers are typically painted on the surface. These markers can be broken into three categories: complex surface gradients [Pritchard and Heidrich 03, Scholz and Magnor 04, Hasler et al 06] (typically detected using SIFT descriptors [Lowe 04]), the intersection of lines [Tanie et al 05] and regions of constant color [Guskov and Zhukov 02, Guskov et al 03, Scholz et al 05]. Our work falls in the third category: regions of constant color. Examining the results in each paper, we observe that constant color markers perform the best. We evaluate previous work by examining the quality of the reconstructed cloth. The most common errors are marker correspondence and are observable in reconstructions by local strain in the reconstructed surface. These errors are noticeable in both still images and video.

[Pritchard and Heidrich 03] used cloth with unique line drawings as markers. Their work identifies parameterization as one of the key aspects of cloth capture. They use a stereo camera to acquire 3D and SIFT descriptors to establish correspondence. Their descriptors are often confused and require significant pruning. They introduce a rudimentary strain metric, as measured along the surface, to rule out incorrect matches. While successful, their static reconstructions show numerous correspondence errors.

[Guskov et al 03] introduce markers of constant color and significantly reduce correspondence errors. In addition, they implement a real-time system and release results on video. Their system, which uses a Kalman filter to smooth, is heavily damped and shows signs of correspondence errors (albeit significantly fewer than Pritchard et al). Their method is limited to simple geometry by the complexity of their color pattern.

[Scholz et al 05] advance [Guskov et al 03] by creating a non-repeating grid of color markers. Each marker has five possible colors and all three by three groups are unique. This allows substantially larger sections of cloth and virtually eliminates correspondence errors. They release high quality videos of a human wearing a shirt and a skirt captured using eight 1K x 1K cameras. The visual quality is compelling, however, they limit the range of motion to avoid occlusion (ex: the arms are always held at 90 degrees to the torso). They use thin-plate splines to fill holes.

We make three main contributions: we improve the color pattern and matching procedure to get more information per marker, we introduce strain constraints to simplify correspondence and we create a data driven hole filling technique that splices previously captured cloth into the mesh. As a result, our system is capable of capturing cloth in real circumstances: with folding and occlusion.

Chapter 6

Texture Coordinates in Multiple Views

This chapter is devoted to the problem of estimating texture correspondence between views. This is similar to estimating texture coordinates in a single view (chapter 2) except that more cues are available in the multiview case. Specifically, when multiple views exist, the strain cue is powerful: accurate localization of points and computation of perspective geometry make the matching procedure much more powerful. Now, thousands of distinct markers are relatively easy to identify.

6.1 Analyzing Acquisition Methods

To acquire a 3D point cloud of the surface of a garment, we print a colored pattern on the surface of cloth, sew together a garment and record motion using multiple synchronized cameras. We then reconstruct the 3D location of surface points by detecting

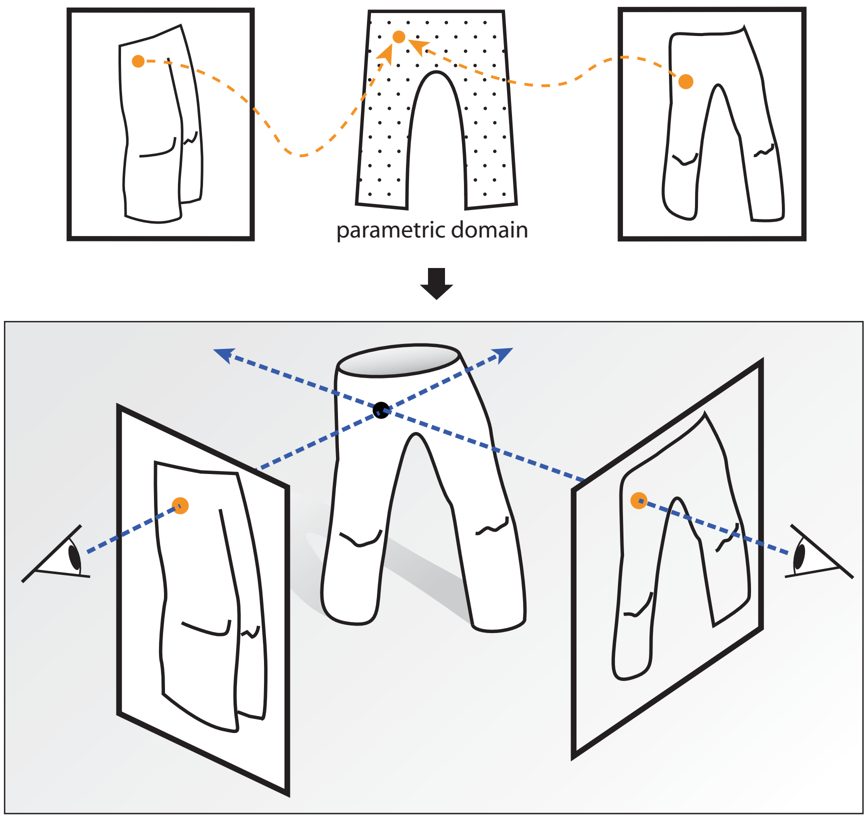


Figure 6.1: Above: We identify corresponding markers in multiple views in reference to the parametric domain. Below: Once corresponding image points are identified, we intersect eye rays to determine the 3D location. In this chapter, we focus on the *correspondence* problem.

corresponding points in multiple views (figure 6.1).

Our goal is high marker density in the 3D reconstruction – especially in regions with high curvature. To achieve this, we need markers that are both small in scale and highly discriminative. These two goals are in tension: small markers are less discriminative. The obvious solution, higher resolution cameras, is limited: camera resolution (or, more accurately, camera bandwidth) is expensive. As a result, we opt for the smallest markers that we can reliably detect and we *make small markers more distinctive*.

We combine information from three cues to establish correspondence: marker color, neighboring markers and strain constraints in the reconstruction. Marker color and strain constraints are more useful than neighboring markers because they place fewer requirements on local cloth geometry. Specifically, neighboring markers are observed only when the cloth is relatively flat. When the surface is heavily curved only small portions of the surface are visible before the cloth curves out of view. In subsequent sections we adopt the following strategy: maximize information obtained from marker color and *eliminate* the information needed from neighbors.

6.1.1 Entropy as an Analytical Tool

We optimize our correspondence technique by analyzing the information provided by different cues. In this framework we can accurately minimize the number of neighbors required for correspondence and observe folds better. We can compare our work to previous methods using this framework (figure 6.3).

It takes $\log_2 M$ bits to determine the identity of each observed marker on a garment

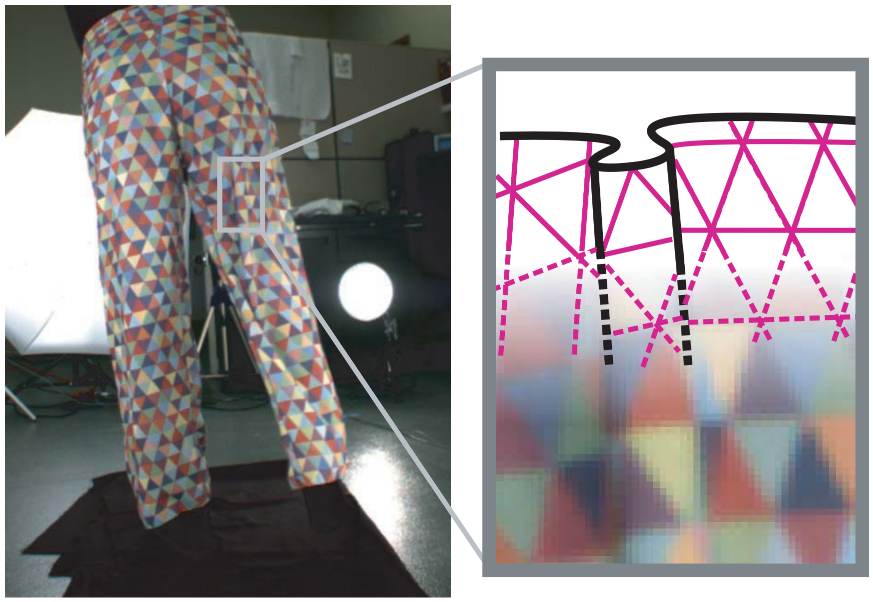


Figure 6.2: Neighborhood detection methods require that all markers in a fixed geometric pattern in the image neighborhood be neighbors on the cloth. Occluding contours break up neighborhood regions and limit the effectiveness of neighborhood methods in folded regions. We eliminate neighborhood requirements in the final stage of our correspondence algorithm.

with M total markers. Because independent information adds linearly, we can compute the information needed to meet this threshold by adding information from the different cues: color, neighbors and strain. However, structural ambiguities in the pattern subtract information lost to determine which neighbor is which. As a result, we compute our information budget ($\mathcal{I}$) as:

$$\begin{aligned} N &= \text{number of observed neighbors} \\ C &= \text{color information per marker} \\ A &= \text{information lost to structural ambiguities} \\ S &= \text{information gained from strain constraints} \\ \mathcal{I} &= (N+1) * C + S - A \end{aligned}$$

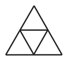
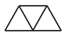
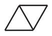
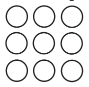
	1st iteration 	2nd iteration 	3rd iteration 	4th iteration	[Scholz 2005] 
Relative Area	15.8	11.8	7.9	4.0	100
Color (C)	≥ 5	≥ 5	≥ 5	≥ 5	1.93
Neighbors (N)	3	2	1	0	8
Strain (S)	0	~ 7	~ 9	~ 9	–
Ambiguities (A)	1.6	1.6	1.6	0	3
Total bits ($\mathcal{I}$)	**18.4**	**20.4**	**17.4**	**14**	**14.4**

Figure 6.3: Our correspondence algorithm iterates from large to small regions. At each stage, the number of recovered bits must stay above the marker complexity (11.6 bits for our pants). We are able to obtain significantly more information per unit cloth surface area than previous work. See section 6.1.1 for the entropy equation and appendix 6.2 for detailed analysis.

As an example, imagine a rectangular grid of markers and a correspondence method that uses a single immediate neighbor. This neighbor is one of four possible neighbors – thus it takes two bits to specify which neighbor we found ($A = 2$). In this case, the equation reduces to $\mathcal{I} = 2 * C - 2 + S$.

Given almost any structured pattern, we can detect regions by increasing N until $\mathcal{I} > \log_2(M)$ bits. However, larger marker regions have the disadvantage that curvature can cause local occlusions and prevent observation of the entire region. Our best efforts are to improve C – the number of bits from each marker observation. We do this by picking marker color from the full colorspace instead of a small discrete set of colors.

6.2 Entropy Comparison

For more reading on information theory, consult [Cover and Thomas 91]. Our analysis is based on the information entropy definition: $H(X) = -\sum_{i=1}^{n} p(x_i) \cdot \log_2 x_i$.

For [Scholz et al 05], the equation in section 6.1.1 is reduced to $\mathcal{I} = 9 * C - A$ because they use 8 neighbors and no strain constraints. They use 5 colors which, without errors, yields $C = \log_2 5$ bits per marker. They cite an error rate of five to ten percent. As a result, they recover anywhere from 1.65 to 2.04 bits per marker. In our comparison, we use $C = 1.93$ bits for color information from their method (five percent error, with equal probabilities for all remaining choices). Note that this is effectively less than four colors! Second, we compute structural ambiguities in their method which account for uncertainty in observations. The orientation of the surface is unknown, yielding four possible directions, or two bits of structural ambiguity. Second, in their paper, they say that oblique views cause another bit of uncertainty. As a result $A = 3$ bits.

For our work, C is an empirical observation. Depending on the lighting and camera configuration, we get anywhere from 5 to 7 bits. We use the conservative estimate of $C = 5$ bits per marker. Second, our mesh is triangular and there are three possible neighborhood rotations, yielding $A = \log_2 3 = 1.59$ bits of structural ambiguity. When neighborhoods are not used, there is no structural ambiguity. Strain information is difficult to compute and depends on the geometry of the surface and the orientation of the camera. In most cases, we observe more than 9 bits of strain information.

6.3 Garment Design and Color Processing

We print a random colored pattern on the surface of cloth in an attempt to maximize the information available per pixel. While our pattern is composed of tessellated triangles (figure 6.2), any shape that tiles the plane will work (squares and hexagons are also natural choices). To maximize the density of reconstructed points, we print the smallest markers that we can reliably detect. To maximize the information contained in the color of each marker, we print colors that span the gamut of the printer-camera response, then use a Gaussian color model (section 6.4.1).

From a system view, the printer-camera response is a sequence of lossy steps: we generate a color image on a computer, send the image to the printer, pose the cloth, and capture it with a camera. Our experiments suggest that loss is largely attributable to camera response because larger markers produced substantially more information. Illumination is also problematic and takes two forms: direct illumination on a lambertian surface and indirect illumination. To correct for variations in direct illumination, we remove the luminosity component from our color modelling. We do not correct for indirect illumination.

Each marker in the printed pattern has a randomly chosen color, subject to the constraint that neighboring marker colors must be dissimilar. In the recognition stage, we detect markers by comparing colors to a known color. These comparisons must be made in the proper color space: we photograph the surface of the printed cloth with our video cameras to minimize the effect of non-linearities in the printing process.

6.3.1 Image Processing

We do some pre-processing to get marker locations and connectivity from raw images. We recommend readers unfamiliar with these techniques refer to [Forsyth and Ponce 02]. We start by converting each image to HSV, disregarding the luminosity (V) and using polar coordinates to compute distances in hue and saturation. To detect markers, our code looks for uniformly colored blobs in two stages: first regions are built by growing neighborhoods based on similarity between pixels. This method is sensitive to image noise and can produce oversized regions when the color boundaries are smoothed. The second stage takes the center of mass of each blob from the first stage, computes the mean color and grows a region based on distance to the mean color (it is computationally intractable to use this as the first stage of the blob detection). The process is iterated for increasing thresholds on the affinity value in the first stage, using the portions of the image where detection failed in previous stages. Finally, blobs are thresholded based on size.

Next, we need to determine the neighborhood relationships. For each marker, we construct a covariate neighborhood (a fitted ellipse) and vote for links to the three closest markers with similar covariate neighborhoods. This measures distances appropriately in parts of the scene where the cloth is receding from view and discourages links between markers with wildly different tilts. All links that receive two votes (one from either side) are kept while the rest are discarded. Links that bridge markers with conflicting color information are also discarded (typically on internal silhouettes).

6.4 Acquisition

The goal of our acquisition pipeline is to compute correspondence using minimal neighborhoods. We accomplish this through an iterative algorithm where we alternate between computing correspondence and pruning bad matches based on those correspondences. After each iteration we shrink the size of the neighborhood used to match. We start with $N = 3$ and end with $N = 0$. In the final iteration, markers are matched using color and strain alone.

This iterative approach allows us to match without neighborhoods. This is better than label propagation methods. To be successful, propagation methods [Guskov et al 03, Scholz et al 05, Lin and Liu 06b] require large sections of unoccluded cloth and must stop at occluding contours. As shown in figure 6.2, occluding contours are both common and difficult to detect. In contrast, our iterative approach relies on strain constraints – which require computing the distance between a point and a line, and color detection – which requires averaging color within a marker. Both of these computations are easier than detecting occluding contours.

We describe our acquisition pipeline, shown in figure 5.1, below.

Color Processing: We compare observed colors with stored values using a Gaussian noise model. Our Gaussian noise model has a single free parameter, the variance, which must be computed empirically for each recording setup. This variance determines the color response for the entire setup — smaller variances mean more bits from color. At this stage, we compute color information for each marker and eliminate hypothetical correspondences from further consideration that have large color differences.

Neighborhood Matching: At each iteration, we match highly distinctive neighborhoods by combining information across cues. The size of the neighborhood is chosen so that we get more than enough bits to meet our information budget ($\log_2 M$ bits – typically 11 to 13). The analysis in figure 6.3 shows that we can set $N = 3$ at the start and continue until $N = 0$. Because the identity of the marker is overspecified, there are few mistakes.

This approach works from flat regions in the first iteration to foldy regions in the

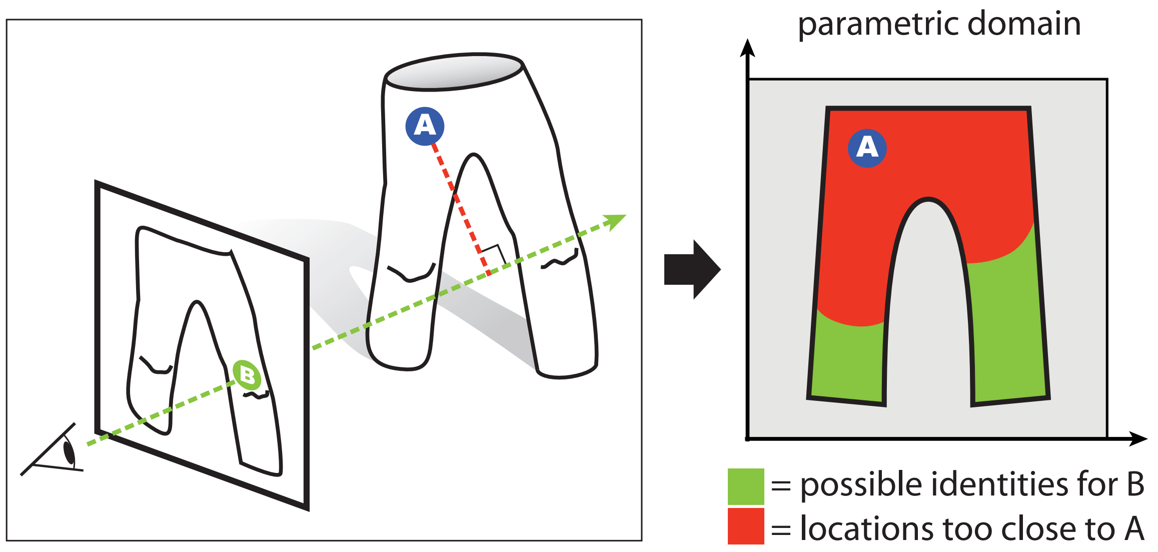


Figure 6.4: **Top**: we compute the shortest distance between a known point A and the eye ray through unidentified image point B. **Bottom**: in the parametric domain, this distance restricts the possible identities of B to the green region. The distance from A to B along the surface can be no shorter than the shortest distance in 3D.

later iterations. In the first iteration, we require three neighbors to make a match. In heavily folded regions, often not all three neighbors are visible. As such, these regions are not going to match. In contrast, in the last iteration, no neighbors are necessary. Occluding contours, which are common in heavily folded regions, no longer disrupt the matching procedure.

3D Reconstruction: Markers that are observed in multiple views (at least 2) are reconstructed in 3D using textbook methods [Hartley and Zisserman 00]. We use reprojection error to prune bad matches (reprojection errors average 0.3 pixels and we discard points with errors larger than 2 pixels).

Pruning with Strain: We do two separate strain pruning steps: one on reconstructed 3D points and one on marker observations in each image. The first discards reconstructed points that cause physically unrealistic strain on the surface of the mesh and the second constrains our search for correspondence. Our strain constraint is based on the work of [Provot 95] who noted that strain in cloth does not exceed 20% in practice. Relaxing the constraint to distances in 3D (surface distance is always less than the distance in 3D), we can use strain to exclude possible correspondences. Strain naturally fits in to our information theory framework: if strain excludes 87.5% of the possible correspondences, then strain has added 3 bits (because $\log_2(1 - 0.875) = -3$). The strain cue is described in figure 6.4.

6.4.1 Representation

To find correspondence, we match each image marker to a marker in the parametric domain. To do this, we define affinities $a_{i,j}$ between image marker i and parametric marker j. Each affinity is a product over different cues. We write $c_{i,j} \in [0, 1]$ for the color affinity, $d(C_i, C_j)$ for the color distance between i and j, $s_{i,j} \in \{0, 1\}$ for the strain constraint, n_i for the image neighbors of marker i and N_j for the parametric neighbors of marker j:

$$\begin{aligned} a_{i,j} &= c_{i,j}\ s_{i,j} \prod_{l \in N_j} \max_{k \in n_i} c_{k,l} \\ c_{i,j} &= \exp\left(-\frac{d(C_i, C_j)^2}{2\,\sigma^2}\right) \\ s_{i,j} &= \begin{cases} 0 & \text{if a strain constraint is violated} \\ 1 & \text{if not} \end{cases} \end{aligned}$$

When only one affinity for image marker i is above a threshold, then we declare a correspondence. Initially, we learned this threshold from labelled data, but we found that changing it by several orders of magnitude had little effect on our results. Subsequently, we use the value $10^{-5(N+1)}$ where N is the number of neighbors.

6.4.2 Capture Results

Our capture results are best evaluated by looking at our video and figures 6.5,8.4,6.6,6.7,6.8. However, to compare against other capture techniques, it is also necessary to evaluate on several numerical criteria. Below, we include a table of numerical data related to each capture session.

	cloth drop	pants dance	table cloth	sleeve†
# cameras	6	8	18	10
resolution	640x480	640x480	900x600	1500x1000
total markers	853	3060	4793	13465
recovered	819	2405	4361	7557
percentage	96%	79%	91%	56%
bits needed	9.7	11.6	12.2	13.7
color bits	6.1	5.1	6.4	4.5
strain bits	9.1	9.4	11.4	~ 6.6

†The sleeve example is unique because it was one of the first items we captured. Much of the cloth is in contact with the floor and unobservable – yielding fewer bits of strain. In addition, the camera images were not output in a linear color space, reducing the number of color bits. As a result, we terminated the correspondence algorithm at $N = 2$.

Our pants animation is by far the most challenging, and we analyze some of the details a little more closely. With an average of 2405 observed markers, there were 979 3D markers per megapixel. If we factor out the pixels lost to background, we get 3500 3D markers per foreground megapixel or 282 foreground pixels per recovered 3D marker. Our marker observations average 56 pixels per marker per image. There are several reasons for the discrepancy: markers must be observed multiple times (approx 44% of 3D markers are observed in 3 or more views), some markers are observed but not reconstructed (due to errors or missing correspondence), and many pixels are not considered part of a marker:

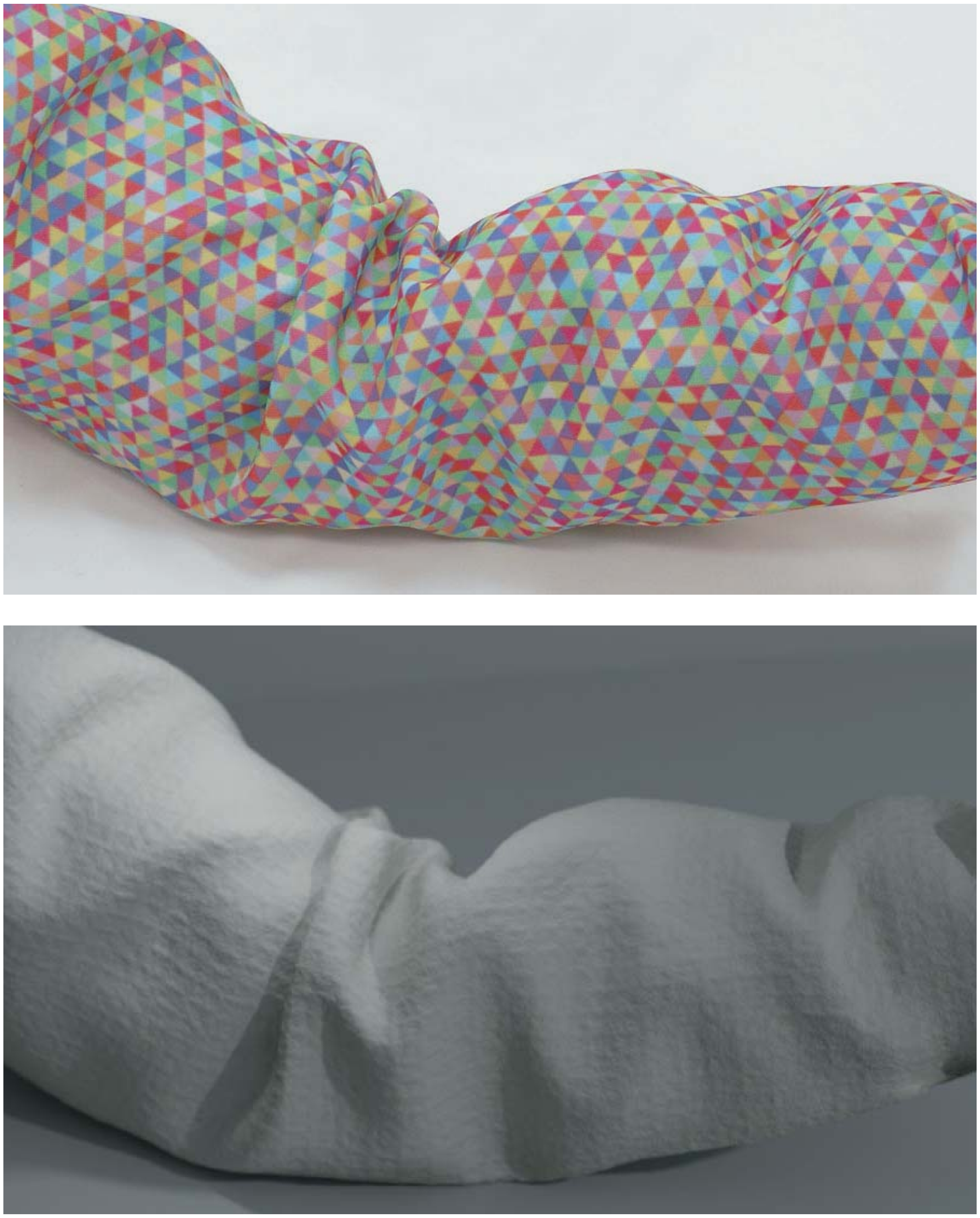


Figure 6.5: We reconstruct the shape of a static arm using thousands of markers to estimate the geometry.

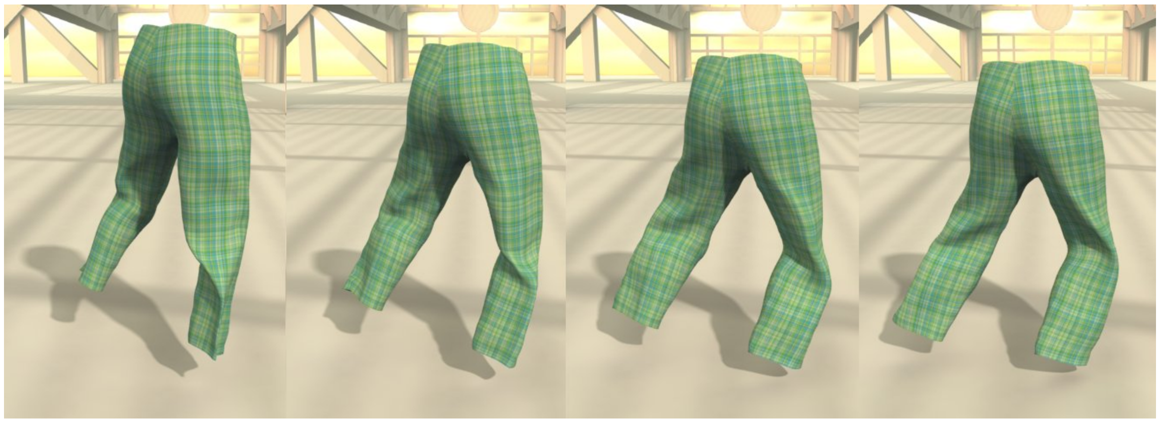


Figure 6.6: We reconstruct cloth being a pair of pants in the process of jumping.

they may lie in a heavy shadow or occupy the edge between two markers (approx 35% of pixels).

6.4.3 Meshing and Cleanup

Our meshes have many problems: there are many markers that are not observed and there is noise in the measurements. However, the most basic problem is that our output is a point cloud instead of a mesh. In section 6.4.3 we discuss the process we use to make a mesh out of the point cloud. In chapter 8, we show how to fill in holes using examples taken from other frames. Finally, in section 6.4.3 we discuss how to smooth the resulting mesh animation to correct for noise in the observations.

Meshing

We produce a mesh by forming equilateral triangles for sections of cloth that are printed with a contiguous pattern by referencing the triangle structure of markers on the

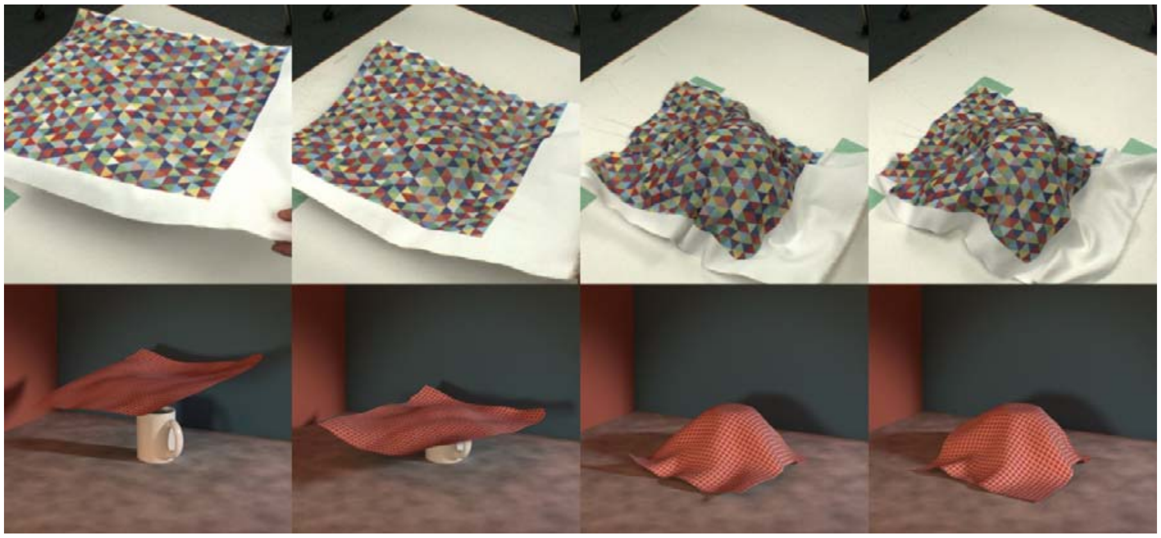


Figure 6.7: We reconstruct cloth being tossed over a cup. The top row shows the original capture data from one of the six cameras and the bottom row shows the resulting reconstruction.

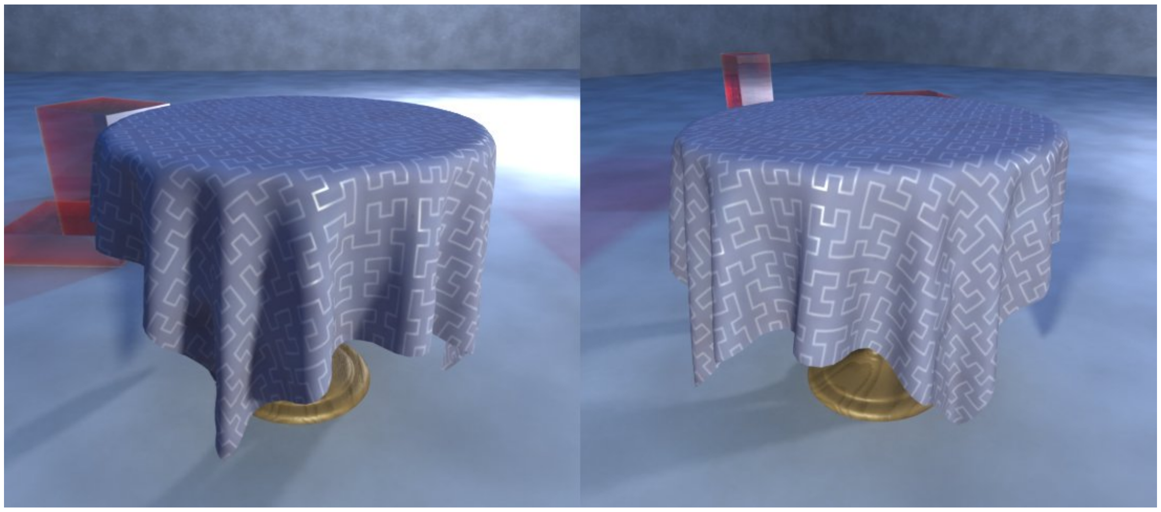


Figure 6.8: We reconstruct a static tablecloth from different viewpoints.

cloth. Our recovered markers are at the center of each triangle – so we average points to get out the vertices and subsequently the original mesh. We insert artificial points where two pieces of fabric come together. These points are created once per garment by hand clicking on photos of the each seam. The 3D locations of these points are recreated in each frame by averaging points near the seam.

Smoothing

We introduce flexibility preserving smoothing – a method similar to anisotropic diffusion [Perona and Malik 90] that smoothes near-rigid movement without effecting flexible deformation. Typical temporal smoothing is dangerous because fast non-rigid movements can easily become physically implausible when blurred over time. However, because fast non-rigid regions of the cloth are complex, small temporal errors are often difficult to notice. In contrast, small errors in regions of the cloth that move rigidly are typically easy to observe. As a result we use flexibility preserving smoothing, a procedure that smoothes rigid movement more heavily than non-rigid movement. To do this, we take a local region around each vertex in the mesh (typically 25 points) and compute a rigid transformation to previous and subsequent frames. Aligning the regions with this transformation, we compute the movement of the vertices in this reference frame as a proxy for rigidity. Large variations in location indicate non-rigid movement and consequently receive little smoothing. Smaller variations indicates rigid movement and benefit from more substantial smoothing. As a result, we can use a Gaussian to smooth in this reference frame.

Chapter 7

3D Geometry

This chapter is a collection of different multiview geometry tools that are useful in cloth capture. Section 7.3 defines a new method for calibrating from deforming objects in multiple views. This method is novel because it uses normals to calibrate and works in the opposite form of most calibration methods: it starts with an orthographic calibration and then works to a perspective calibration. Normals are a particularly powerful tool for cloth because they can be easily computed from the deformation of surface textures.

Then, in section 7.4 we look at the problem of reconstruction. Here, we assume that correspondence and calibration are both solved. Then, the question becomes: what is the best way to reconstruct the 3D points that correspond to the surface of the cloth. We use the notion of strain to define a novel form of bundle adjustment to produce more accurate reconstructions of the cloth surface. Finally, we look at volume cues and specifically the extraction of silhouette edges to aid in reconstruction where data is hard to observe. In both this chapter and chapter 8, we observe that occlusion is a fact of cloth capture. In this chapter, we use volume cues to help fill in missing data. In chater 8, we use previous observations of cloth to fill in missing data.

7.1 Background

7.1.1 Structure-from-Motion

Camera calibration is well understood, with comprehensive reviews in two excellent books [Faugeras and Luong 04, Hartley and Zisserman 00]. Software that implements the most common techniques is readily available on the Internet [Bouguet 05]. However, these tools have drawbacks when used to capture dynamic cloth, which moves through large volumes while showing minimal perspective effects. We begin with a review of the relevant terminology.

Camera parameters are described as a set of **extrinsic** (configuration) and **intrinsic** (focal length, camera center, etc.) variables. Camera calibration (determining the camera parameters) can be broken into two categories: **photogrammetric calibration**, where the geometry of the scene is known ahead of time to high precision; and **auto-calibration** where the structure of the scene is not known ahead of time and is simultaneously recovered from the 2D views.

The standard method involves: identifying **interest points**; using **appearance**, **epipolar** and **three view** constraints to build frame-frame correspondences between these points; obtaining a **projective reconstruction** — which yields geometry and cameras up to a 3D projective transformation — using one of several current factorization methods; and then using appropriate assumptions to obtain an **upgrade** to a Euclidean reconstruction.

The reconstruction and cameras are then cleaned up with a **bundle adjustment**, which minimizes reprojection error as a function of reconstruction and camera parameters.

While we do not directly assume the geometry of the scene, our work fits in with previous work in calibration assuming a calibration object. Early work in this area used a planar object with vertices at known locations [Tsai 92]. More recently, improvements to the solution and readily available executables have made planar calibration commonplace [Bouguet 05, Zhang 00]. In contrast to these approaches, we make no assumptions about the scene geometry, but instead assume that normals can be computed from our fixed pattern. Our cloth pattern is composed of triangles at multiple scales and can be seen in figure 2.2.

Our application is cloth motion capture. As an example of the common approach to calibration, [Scholz et al 05] use a checkerboard pattern painted on the floor of the studio to calibrate the cameras — constraining the location and motion of the cameras while consuming a large number of valuable pixels.

7.2 Texture Geometry in Multiple Views

Reconstruction from scaled orthographic views is now a standard algorithm (originating in [Tomasi and Kanade 92]; many important variants appear in [Hartley and Zisserman 00]). If there are more than two cameras, a metric reconstruction is available by enforcing scale and angle properties of the camera basis. However, this approach ignores knowledge of scene geometry (because we know what a triangle looks like, we can estimate a surface normal). A metric reconstruction isn't possible from two views in simple orthographic cameras without calibration of camera extrinsics or some known length or an-

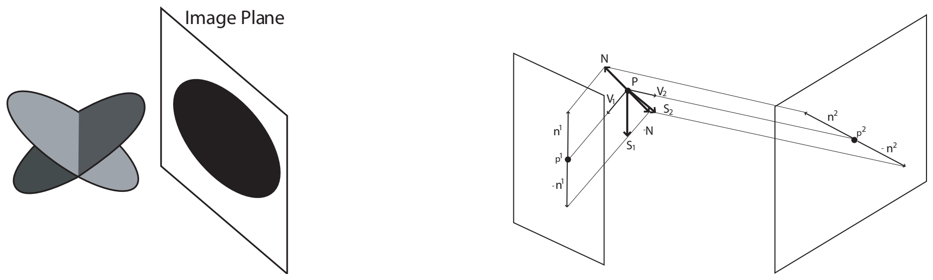


Figure 7.1: **(Left)** Viewing a planar circle yields an ambiguous image of an ellipse in the image plane. Two possible circles correspond to the ellipse in the image. **(Right)** Notation for normal ambiguity in two views. There are two simple orthographic views of the point $\mathbf{P}$, with normal $\mathbf{N}$; view directions are $\mathbf{V}_1$ and $\mathbf{V}_2$. The text shows that $\mathbf{S}_1$ and $\mathbf{S}_2$ — ambiguous normals in their respective views — have heads lying on the same epipolar plane and that an incorrect match leads to a reconstruction of $-\mathbf{N}$.

gle [Huang and Lee 89, Ullman 79]. Since the cloth moves fast and we may be stuck with only two views, and to incorporate our normal information, we adopt a method that exploits surface normals to obtain a metric reconstruction.

In a single scaled orthographic view, we know the normal of the plane on which the pattern element lies *up to a two-fold ambiguity* (e.g. [Forsyth 02, Lobay and Forsyth 04]). This ambiguity occurs because we can identify the cosine of the slant angle — usually written as $\cos\sigma$ — but not its value from a single view. For example, a scaled-orthographic view of a circle looks like an ellipse; we know the extent of the slant (and so the length of the normal) but the circle could have been slanted either forward or backward to yield this ellipse (figure 7.1). As a result, we know the projected normal up to an ambiguity of π radians.

The most natural way to incorporate this information into existing multiple view results is to think of the normal as an arrow of known length protruding from the surface at the point in question. The base of the arrow is the point in question, and projects as usual. The results above mean we know (up to a two-fold ambiguity) to what point in the image the head of the vector projects — in turn, having a normal from texture repetition is equivalent to having a second point *and* having some metric information *because we know the length of the normal vector.* For convenience, in what follows we refer to an isolated point as a **point**, and a point with the normal information described as a **patch**.

7.2.1 The 3D Ambiguity of Normals

Assume that we are dealing with a pair of simple orthographic cameras. Furthermore, assume that the scale of the cameras is the same (we can obtain the relative scale from the size estimates for triangles), and that the extrinsics are calibrated. In a single view, the projected normal is known up to an ambiguity of π radians. What ambiguity is there in 3D reconstruction of the normal?

Write the normal as $\mathbf{N}$ and the i'th view vector pointing toward the camera (figure 7.1) as $\mathbf{V}_i$. In the i'th view, there are two possible 3D normals, $\mathbf{N}$ and $\mathbf{S}_i$ (the ambiguous normal in the i'th view). Because the image ambiguity is π radians, $\mathbf{N}$, $\mathbf{V}_i$ and $\mathbf{S}_i$ must be coplanar. Because the projected length of $\mathbf{S}_i$ is the same as the projected length of $\mathbf{N}$, $\mathbf{V}_i \cdot \mathbf{N} = \mathbf{V}_i \cdot \mathbf{S}_i$. This means that we have $\mathbf{S}_i = 2(\mathbf{N} \cdot \mathbf{V}_i)\mathbf{V}_i - \mathbf{N}$ The epipolar planes consist of every plane whose normal is $\mathbf{E} = \mathbf{V}_1 \times \mathbf{V}_2$. The "heads" of $\mathbf{S}_1$ and $\mathbf{S}_2$ lie on the same epipolar plane, because $\mathbf{E} \cdot \mathbf{S}_1 = \mathbf{E} \cdot \mathbf{S}_2 = -\mathbf{E} \cdot \mathbf{N}$. In the circumstances described, there are two possible matches for the "head" of the normal. First, the correct matches are available, resulting in a reconstruction of $\mathbf{N}$; second, one can match the image of the "head" of $\mathbf{S}_1$ with the image of the "head" of $\mathbf{S}_2$. The second case results in a reconstruction of $-\mathbf{N}$

(figure 7.1); this is easily dealt with, because visibility constraints mean that $-\mathbf{N} \cdot \mathbf{V}_i < 0$ for both i.

All this yields **Lemma:** *A metric reconstruction from two simple orthographic views is available from two patch correspondences. There is a maximum of sixteen ambiguous cases, yielding no more than four camera reconstructions.* **Proof:** (see [White and Forsyth 05]) There is an obvious **corollary:** *A fundamental matrix is available from two patch correspondences, up to at worst a four-fold ambiguity.*

7.3 Calibration with Deforming Textured Objects

In this section, we develop a method to calibrate perspective cameras from views of a deforming textured object. This is particularly useful because it allows us to calibrate our recording setup from observations of cloth alone – giving us flexibility to capture in new environments without calibrating ahead of time. With digital still cameras, this is particularly useful because images can be taken without a tripod or calibration object.

We proceed by finding an orthographic calibration, then enriching the camera model to include perspective effects and distortion terms. In the process, we establish the utility of using surface normals to calibrate cameras in the orthographic setting, with a proof that a metric reconstruction can be achieved in two orthographic views of two points combined with two normals. Our calibration object is a sheet of textured cloth. We show that a calibration applies to a specific region of space. Substantial improvements in reconstruction quality and in reprojection error can be obtained by using calibration objects that explore most of the relevant 3D volume. Using cloth as a calibration object allows easy calibration of large 3D volumes more accurately than typical planar calibration objects.

7.3.1 Overview

We present a new method of camera calibration targeting the reconstruction of cloth in moving sequences. Our requirements differ from common structure from motion estimation problems: we typically have a small number of cameras surrounding a roughly convex textured object, each point is typically viewed by three or fewer cameras, perspective effects are small but not negligible and neighboring points may be reconstructed from different cameras. As a result, small calibration errors result in large strains in the reconstructed cloth — a visually displeasing and physically implausible artifact.

Our main claim is that using a 'standard' fixed planar calibration object (a checkerboard in the bottom of the scene — see figure 9 in [Scholz et al 05]) is not ideally suited for this application. First, this fixed object places unnecessary restrictions on the locations of the cameras. Even worse, when all cameras can view the same plane, this approach wastes resolution in the capture system. Second, we demonstrate that calibration is specific to a volume of space. Self calibration methods (such as [Pollefeys et al 99]) obtain good reconstructions because they use the same correspondences for both calibration and reconstruction. To achieve similar results from a secondary calibration pattern, one would need to move the calibration object through the volume before recording. However, it is difficult to guarantee that the appropriate volume in 3D is covered effectively. We opt for a more convenient technique: build the calibration into the deforming object, which then effectively explores the space. We calibrate over time — using the large numbers of correspondences to calibrate and thus guarantee that our calibration pattern covers the relevant volume.

It is not intuitive that one could calibrate cameras with a deforming object. In fact, as we show, the advantages of exploring the view volume are substantial. Our reprojection errors are significantly lower (figure 7.7) and, in contrast to other techniques, our reconstructions agree with physical cloth models. (figure 7.8)

Our calibration method differs from previous approaches. Observing that perspective effects are small, we calibrate in two steps: first, we compute an orthographic camera calibration using the printed cloth pattern, then 'enrich' the model to include perspective effects and distortion parameters. While general orthographic calibration of point clouds requires 3 views of a sufficient number of points to get a metric reconstruction, we prove in the appendix that using points and normals, a metric reconstruction can be obtained from only two orthographic views if two corresponding points and normals are known. While an orthographic view of any repeated pattern can provide normals [Lobay and Forsyth 04], in this work we consider the case where we know the frontal texture pattern *a priori.*

7.3.2 Orthographic Camera from Two Views

To obtain a camera solution from two views, we perform a rough search over rotation matrices, then use gradient descent to refine the solution. Our cost function for this is based on the combination of two penalties: the reprojection cost of the points (in pixels) and the alignment cost of the normals (in degrees). Using $\mathbf{x}_i^j$ as the observed point i in view j, $\hat{\mathbf{x}}_i^j$ as the reprojected points, $\mathbf{N}_i^j$ as the respective normals and $\mathcal{R}$ as the rotation matrix between the two views, our costs can be computed as follows:

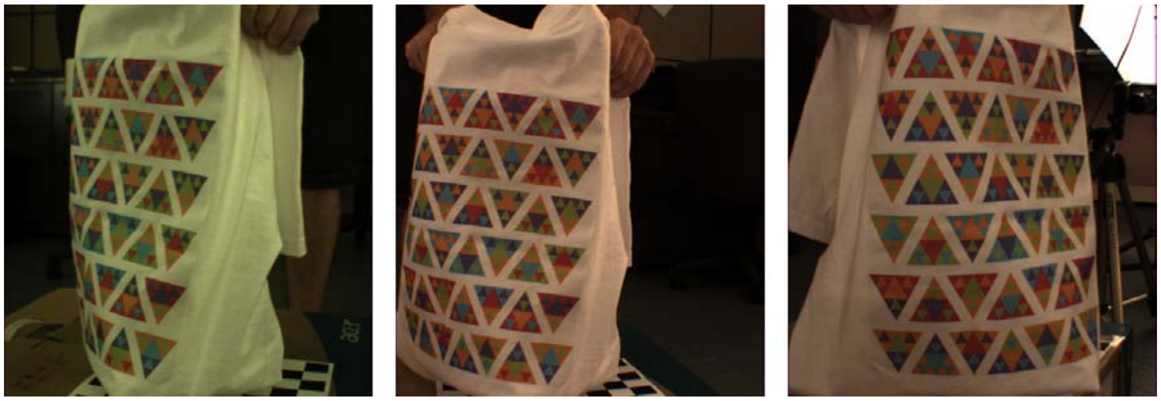


Figure 7.2: Perspective effects of cloth are typically small. As a result, we can ignore them at first, obtain an orthographic reconstruction, then solve for a reconstruction from a perspective model. In these frames, we start with pairwise orthographic calibrations computed over a sequence of frames. Following the technique described in section 7.3.2, we minimize a combined objective that includes reprojection error and agreement in the rotated normals. To compute the agreement of the normals, we use the relative rotation matrix between two cameras to rotate normals from one viewpoint to another and report the angular alignment of the normals to verify the quality of the reconstruction. Averaging over 20 frames of a dynamic sequence viewed by three cameras, we have errors of 3.10 pixels between the first view and the second and 2.96 pixels between the second view and the third.

$$\begin{aligned} c_{\text{normals}} &= \textstyle\sum_i (1 - \mathbf{N}_i^1 \cdot \mathcal{R}\mathbf{N}_i^2) \qquad\qquad c_{\text{pts}}^j = \textstyle\sum_i \|\mathbf{x}_i^j - \hat{\mathbf{x}}_i^j\|^2 \\ c_{\text{pts}} &= c_{\text{pts}}^1 + c_{\text{pts}}^2 + \text{const} \cdot c_{\text{normals}} \end{aligned}$$

To obtain the reprojected reconstructed points $\hat{\mathbf{x}}_i^j$ from the observations and rotation matrix, we first move the center of gravity of the observed points to the origin. Then, we create an orthographic camera matrix $\mathcal{K}$ as $[\mathbf{i}^T, \mathbf{j}^T, \mathbf{r}_1^T, \mathbf{r}_2^T]$, where $\mathbf{i}$, $\mathbf{j}$ are in the coordinate axis directions as usual and $\mathbf{r}_1^T$ and $\mathbf{r}_2^T$ are the first two rows of the camera rotation matrix $\mathcal{R}$. Finally, we compute the reprojected reconstructed points $\hat{\mathbf{x}}_i^j$, by using the pseudo-inverse of $\mathcal{K}$ to reconstruct and $\mathcal{K}$ to reproject. A grid search over rotation matrices $\mathcal{R}$ provides an estimate of the camera matrix and gradient descent refines the parametric camera model. Rotation estimates from this approach

Because normals are estimated from texture elements which effectively display no perspective effects individually (they are too small), our method yields a rotation estimate for *perspective* cameras from the locally valid assumption of scaled orthography. This remarkable fact is borne out by the excellent consistency of our normal estimates and our rotation estimates. In particular, applying our rotation to normal estimates between views yields alignment within two degrees. (Figure 7.2) The accuracy of this alignment also indicates that normals estimates themselves are accurate, even in images with significant wrinkles.

7.3.3 A Perspective Camera from Bundle Adjustment

At this point, we have a structure estimate and an estimate of camera extrinsics and scale assuming scaled orthography. Our cameras may not, in fact, be scaled orthographic cameras, and some lateral views of cloth display mild perspective effects (Figure 7.2). This results in potentially large reprojection errors and poor reconstructions. We use the orthographic camera solutions as an initialization for a fuller perspective model. We then run bundle adjustment: a large minimization over the camera parameters (both intrinsic and extrinsic) and the reconstructed points. We refrain from a complete discussion of bundle adjustment here, and refer readers to [Triggs et al 00] for more details.

7.3.4 An Orthographic Camera in a Perspective Model

To use the orthographic camera calibration as an initialization for the perspective model, we represent the orthographic camera in the richer perspective model.

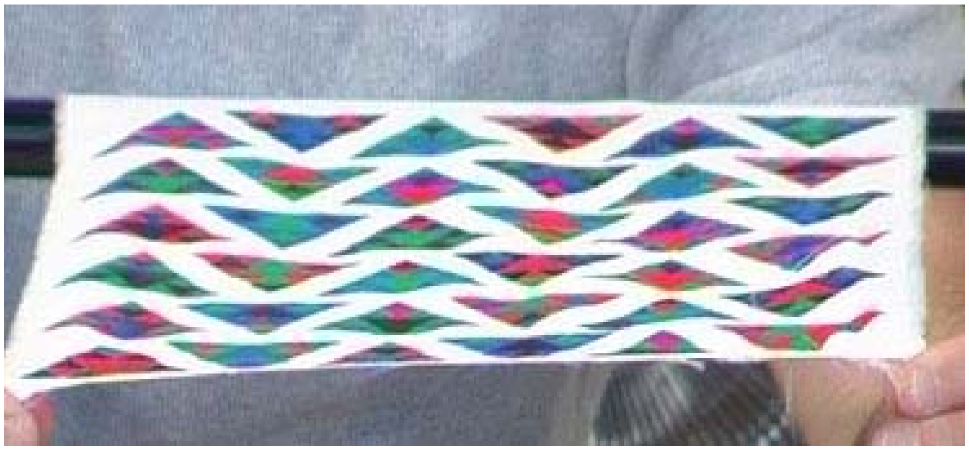


Figure 7.3: Cloth can show minimal perspective effects in some views, typically when the plane of the cloth is nearly perpendicular to the plane of the camera. However, since cloth tends to cover roughly convex objects, views rarely exhibit the same level of perspective effects observed in images of streets, hallways, or building exteriors.

Our camera model is based directly on the model adopted by [Bouguet 05], and is similar to the models used in [Zhang 00, Heikkila and Silven 97]. We will follow the notation of [Zhang 00] as appropriate. To distinguish between the orthographic and perspective equations, we adopt the subscript π for perspective and o for orthographic. Using homogeneous coordinates for the points in the image ($\mathbf{m} = [u, v]^\top$, $\tilde{\mathbf{m}} = [u, v, 1]^\top$) and points in 3D ($M = [X, Y, Z]^\top$, $\tilde{M} = [X, Y, Z, 1]^\top$), we write the basic camera as:

$$\tilde{\mathbf{m}} = \mathbf{A}\ \mathcal{C} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0\,0\,0 & 1 \end{bmatrix} \tilde{M} \qquad \mathbf{A} = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $(\mathbf{R}, \mathbf{t})$ are rotation and translation, the matrix $\mathcal{C}$ is the camera model ($\mathcal{C}^o = [1000; 0100; 0001]$, $\mathcal{C}^\pi = [1000; 0100; 0010]$), (u_0, v_0) are the coordinates of the principal point, and (α, β) are scale factors. To account for distortions due to lens artifacts, we use a 6 DOF model, taken directly from [Bouguet 05] and inspired by [Heikkila and Silven 97].

While our model includes distortions due to lens artifacts, for cleanliness we drop these terms below. To distinguish between the orthographic and perspective equations, we adopt the subscript π for perspective and o for orthographic. Using (u, v) as coordinates in 2D, $(M = [X, Y, Z])$ as coordinates in 3D, $(\mathbf{R}, \mathbf{t})$ as rotation and scale, (u_0, v_0) as the principal point, (α, β) as perspective scale factors and s as the orthographic scale factor, we write the projection along a single camera axis as:

$$u^o = s(\mathbf{r}_1^o M + t_x^o) \qquad\qquad u^\pi = \alpha \tfrac{\mathbf{r}_1^\pi M + t_x^\pi}{\mathbf{r}_3^\pi M + t_z^\pi} + u_0^\pi$$

We note that the equations appear in a fairly similar form. By making the assignment $\alpha = t_z^\pi s$, in the limit of large t_z the perspective model becomes orthographic:

$$\lim_{t_z^\pi \to \inf} u_\pi = s(\mathbf{r}_1^\pi M + t_x^\pi) \left(\lim_{t_z^\pi \to \inf} \frac{t_z^\pi}{\mathbf{r}_3^\pi M + t_z^\pi} \right) + u_0^\pi = s(\mathbf{r}_1^\pi M + t_x^\pi) + u_0^\pi$$

Since the orthographic camera is the limit of the perspective camera, simple substitution allows us to use our orthographic calibration as an initialization for the perspective camera. We start by assigning the rotation matrices to be the same ($\mathbf{R}^\pi = \mathbf{R}^o$). An ambiguity exists in computing t_x^π and u_0^π from t_x^o. We assume that u_0^π should lie near the center of the camera, and use the following equations to complete the transition from orthographic to perspective:

$$\begin{array}{llll} t_z^\pi \to \inf & \alpha = t_z^\pi s & u_0^\pi = \dfrac{\text{image width}}{2} & t_x^\pi = t_x^o - \frac{u_0^\pi}{s} \\ & \beta = t_z^\pi s & v_0^\pi = \dfrac{\text{image height}}{2} & t_y^\pi = t_y^o - \frac{v_0^\pi}{s} \end{array}$$

Substitution in Practice

The only complication in this procedure is the initialization of the parameter t_z. As suggested in the previous section, we should initialize this to almost infinite value. However, in practice, we simply choose a value significantly larger than scene geometry suggests. Second, as outlined in the appendix, an orthographic view has an ambiguity in depth that does not occur in the perspective case. From an orthographic given camera, a flip in the z coordinate of the other camera matrices will produce a depth flipped reconstruction. However, in the perspective case, an erroneous flip in depth would cause nearby objects to appear smaller. To account for this, we search over the two cases for each camera matrix.

7.3.5 Experiments

We implemented the calibration framework established in this paper and printed several pieces of cloth and paper with our triangle pattern. Through experiments with real world data, we establish the following points:

A metric upgrade of a textured surface is available from two orthographic views. We establish this by photographing a scene of known geometry (two planes that form a right angle) and measure the angle from the reconstructed geometry to be within one degree. (Figure 7.4) Our rotation estimates are accurate — implying that normal estimates are highly reliable.

Our calibration code for perspective scenes is comparable to existing methods in standard configurations. We show this by calibrating with a planar calibration object and calibration software available online. (Figure 7.5)

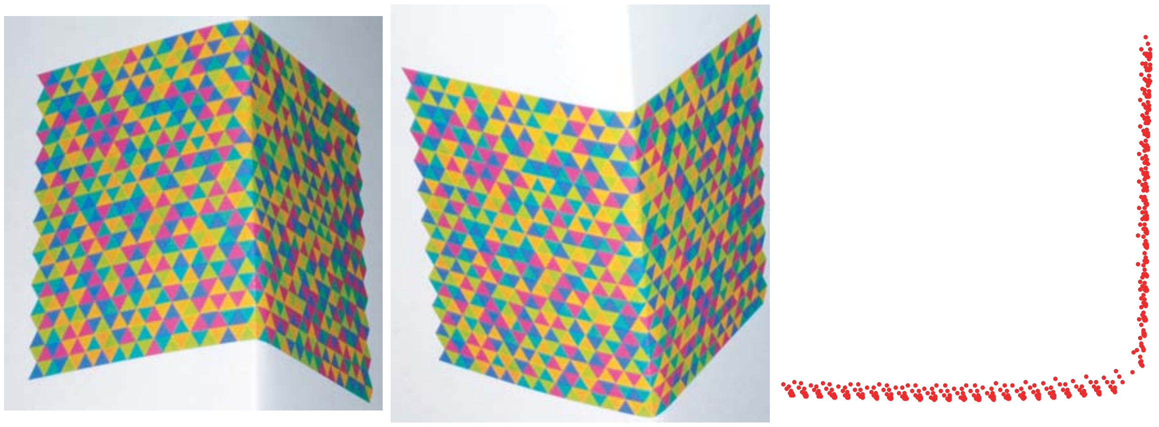
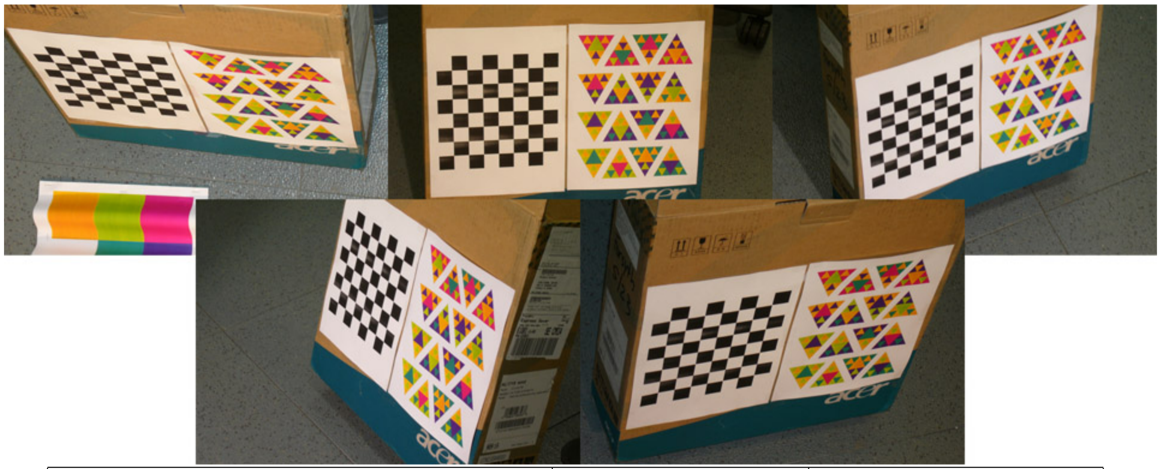


Figure 7.4: We demonstrate the ability to obtain a metric reconstruction from two orthographic views of a textured scene. We paste a triangle pattern on a box, forming two sections at a roughly 90 degree angle. Using both points and normals, we reconstruct the point locations by computing an orthographic camera calibration. Our calibration error is on average 4.94 pixels and 1.07 degrees. (photos taken with a consumer camera at maximum zoom to approximate orthography) To evaluate the results, we fit two planes to the two point sets and compute the angle between the planes to be **88.3 degrees** (we estimate 90 degrees from world geometry). When fitting the plane the MSE distance from the points to the plane is 2.59 pixels. Inspection of the errors appear to include unmodeled perspective effects at the scale of the scene. As a result, reprojection errors are large with respect to state of the art for perspective calibration, but estimation of normals and angles is highly accurate.

Calibration of 3D volumes

Finally, we establish: **calibration is specific to 3D volumes and calibration in the same volume is superior**. Empirically, we observe that calibration objects are better when they occupy the same 3D volume as the measured structure. To obtain better reconstructions, one needs to use a calibration object that is large and centrally located. However, in moving sequences, such calibration objects wastes resolution — occupying valuable pixels that could be used to estimate instead.

Using cloth as a calibration object in the same volume is substantially better than using a traditional calibration object. To gauge the effectiveness of our method we calibrate



calibration data (camera model)	**evaluation data**	**reprojection error**
triangle pattern (orthographic)	triangle pattern	0.995 pixels
triangle pattern (perspective)	triangle pattern	0.318 pixels
triangle pattern (perspective)	checkerboard	3.270 pixels
checkerboard (perspective)	checkerboard	0.172 pixels
checkerboard (perspective)	triangle pattern	1.220 pixels

Figure 7.5: We compare our calibration pattern with standard software available in [Bouguet 05]. Our method does not perform as well, probably because it does not assume fixed geometry — limiting its accuracy in this confined case. Note that both methods generalize poorly, implying that calibration should be performed in the same region as reconstruction. As shown in figure 7.7, when calibrating larger regions of space, this problem becomes even more pronounced. Our calibration method starts by using texture cues to obtain a metric reconstruction for orthographic cameras, then 'enriches' the model to include perspective effects using bundle adjustment.

on one portion of the sequence, and compute errors on another portion of the sequence. However, in practice, calibration should be performed using the entire sequence.

In Figure 7.6, we give example images from a sequence of moving cloth and in Figure 7.7, we show that reprojection errors are significantly worse when using the planar calibration object in one portion of the scene. Even worse, such calibration objects require that every camera is able to see the planar object. This restriction can be limiting in the case of large numbers of cameras and awkward geometries.

Reprojection errors are not just a problem in theory, but also in practice. As figure

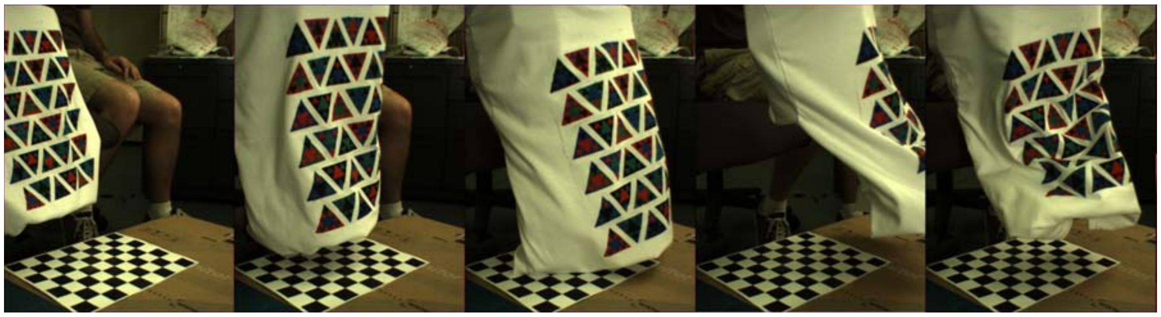


Figure 7.6: Above is a selection of images taken from a single camera — the cloth moves significantly during different parts of the sequence. A planar calibration pattern (checkerboard) allows us to compare calibration methods (checkerboard calibration performed using code from [Bouguet 05]). We calibrate over time using several frames in order to cover the space well. Figure 7.7 shows calibration results for this sequence.

calibration data (camera model)	**evaluation data**	**reprojection error**
cloth: frames 1-10 (orthographic)	cloth: frames 1-10	2.06 pixels
cloth: frames 1-10 (perspective)	cloth: frames 1-10	0.35 pixels
cloth: frames 1-10 (perspective)	**cloth: frames 11-50**	**0.68 pixels**
cloth: frames 1-10 (perspective)	checkerboard	2.15 pixels
checkerboard (perspective)	checkerboard	0.30 pixels
checkerboard (perspective)	**cloth: frames 11-50**	**2.62 pixels**

Figure 7.7: Both calibration objects (the cloth and the checkerboard) give good calibration results on the data used to calibrate and both methods generalize poorly to data in physically different locations. However, the calibration using the cloth gives significantly better results on cloth data from another part of the sequence. On unseen data, calibration from the cloth produces average errors of 0.68 pixels, while calibration from the checkerboard produces average errors of 2.62 pixels. This is because the cloth moves through the same 3D volume in one portion of the sequence, allowing a better reconstruction in the remaining portion of the sequence. The checkerboard calibration pattern has the additional disadvantage that it must be viewed by all cameras and often occupies valuable camera pixels.

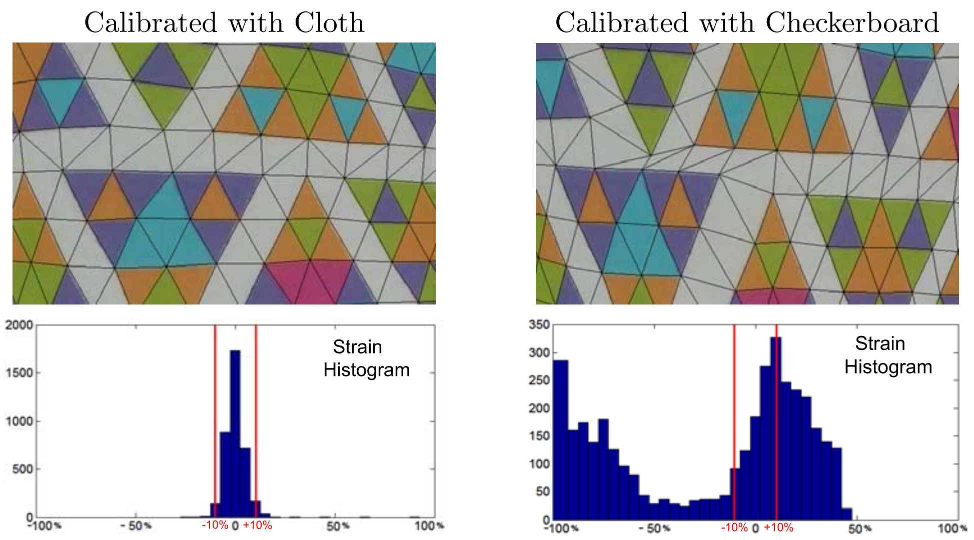


Figure 7.8: We demonstrate the power of our calibration method by computing reconstructions that are consistent with the physics of cloth. On the **left**, a close-up view of a reconstruction using the triangle pattern in other frames to calibrate. On the **right**, a similar view reconstructed using the checkerboard as a calibration object. (original photos of this sequence in figure 7.2) Errors in the checkerboard reconstruction are two-fold: First, the relative scaling of the axes is less accurate. Second, small errors in calibration produce strain — or stretch and compression of the edges connecting neighboring vertices. Notice substantial errors on the right in the white regions surrounding each triangle. **Below**, histograms of the strains over the whole surface highlight the importance of good calibration. Values that deviate significantly from zero (rest length) correspond physically to large forces on the cloth. [Provot 95] notes that strains for textile materials are typically less than 10%. (Technical note: strain is $\frac{\Delta L}{L}$, where L is the rest length. To compute rest lengths, we estimate a scale factor between the model and the reconstruction. For checkerboard calibration, our estimates are dubious because we choose a scale estimate that minimizes strain. When the strains are consistent, this task is easy, but when they are inconsistent, the task becomes harder. No matter what method we use, the checkerboard calibration produces large strains)

7.8 demonstrates, the reconstruction offered by our method is consistent with physical models of cloth. Cloth geometry reconstructed from poor calibration can be heavily strained — meaning that the distance between points is not accurately recovered. In graphics applications, strain is considered so distracting that some simulation methods explicitly contain a strain reduction step [Provot 95].

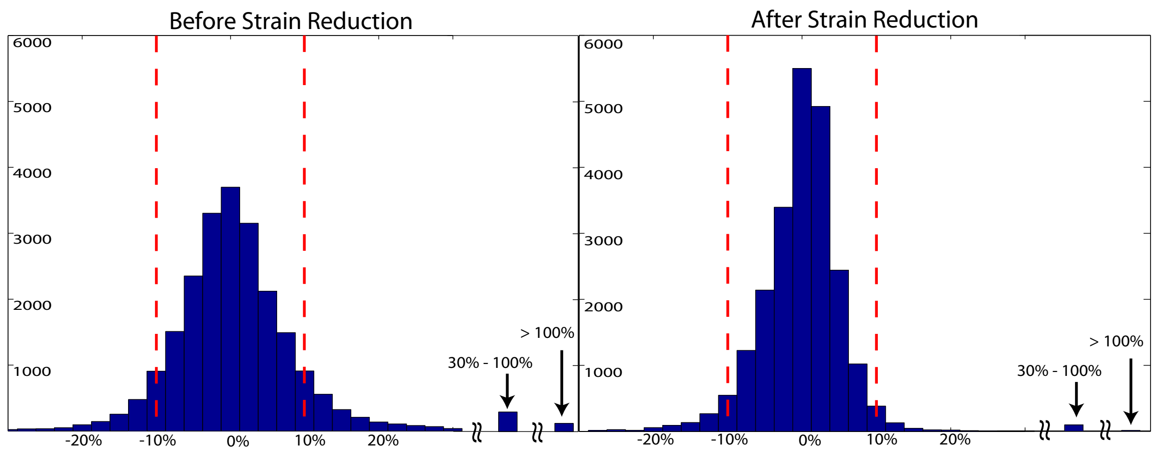


Figure 7.9: Strain reduction dramatically cuts down on strain in the mesh while preserving agreement with the image data. Using the data from figure 1, the reprojection error (the agreement between the model and the image data) increases a small amount (10 μm) but creates a substantial reduction in strain. The red lines indicate a strain of 10% – an amount used in simulation as a typical maximum. We physically measured a maximum strain of 15% at 45° to the grain of this particular polyester cloth by stretching it to the point that there was a concern that the cloth would tear.

7.4 Strain and Bundle Adjustment

Now assume we know correspondence between views of points seen in two or more views (chapter 6 describes how we obtain correspondence) and we have a calibration. Reconstruction is not straightforward, because many points are seen with short baselines. However, we can exploit cloth's resistance to strain to improve the reconstruction.

Standard bundle adjustment proceeds by minimizing the reprojection error to polish both 3D locations of reconstructed points and camera parameters. We improve upon this by penalizing large strains, after an idea due to Provot [Provot 95]. Small strains in cloth result in relatively small forces, but slightly larger strains can produce large forces. Because we have recovered a parameterization, we can observe strains in the recovered cloth model. We create a global cost function that combines the reconstruction error in

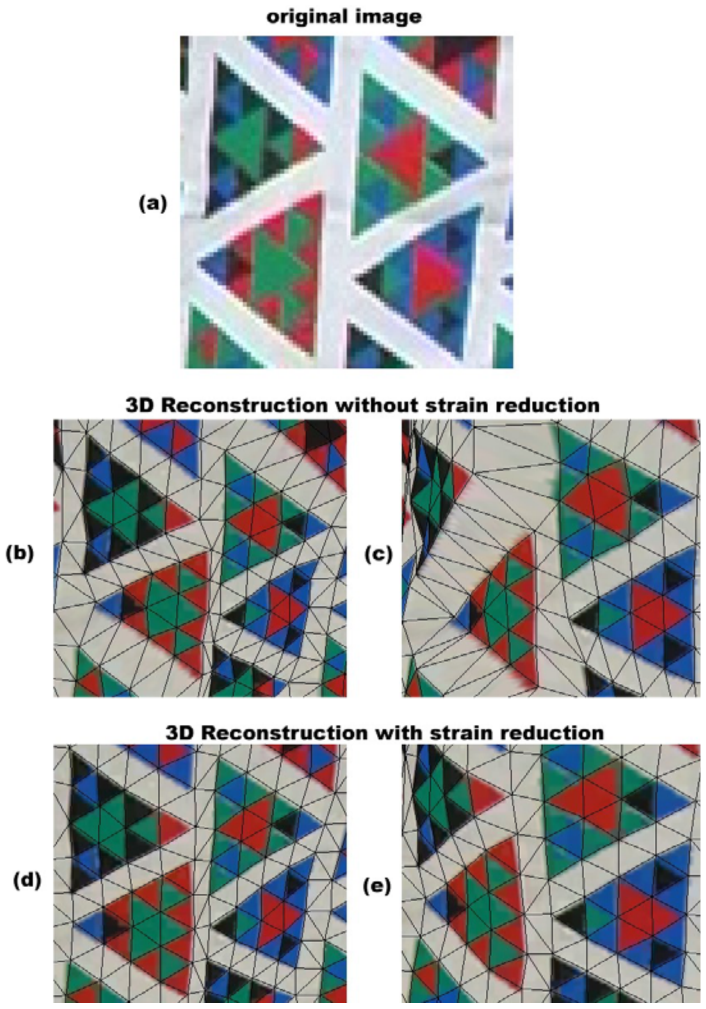

original image
(a)
3D Reconstruction without strain reduction
(b)
(c)
3D Reconstruction with strain reduction
(d)
(e)

Figure 7.10: Because we have a small number of views of each triangle, minimizing the reprojection error alone only produces an accurate mesh when viewed from similar viewpoints. Images **(b)** and **(d)** are **rendered** views of the **reconstructed** mesh (textured with a frontal view of the flattened cloth) taken from viewpoints similar to the original image **(a)**. However, without strain reduction, novel views do not exhibit cloth-like structure. The reconstructed mesh in image **(c)**, produced by minimizing reprojection error alone, is rendered from a view significantly different from all the original cameras. Note that this results in significant variance in the mesh — indicated by large variations in edge length. Image **(e)** shows a similar rendered view of a reconstructed mesh produced by **simultaneously** minimizing reprojection error and strain (section 7.4). Now, the structure of the mesh is more realistic and true to the original image data.

each camera with the strain on the mesh (defined by a Delaunay triangulation of the points in the cloth domain). Using $\|\mathrm{e}\|$ as the edge length, $\|\mathrm{e}_r\|$ as the rest length, $E_r(\mathbf{p})$ as the reconstruction error and k_s as the weight of strain relative to reconstruction error; our cost function is defined as:

$$
\begin{aligned}
\mathrm{strain(e)} &= \begin{cases} (\|\mathrm{e}\| - \|\mathrm{e}_r\|)^2 & \text{if } \|\mathrm{e}\| > \|\mathrm{e}_r\| \\ 0 & \text{otherwise} \end{cases} \\
\mathrm{cost} &= k_s \sum_{\mathrm{e} \in \mathrm{edges}} \mathrm{strain(e)} + \sum_{\mathbf{p} \in \mathrm{points}} E_r(\mathbf{p})
\end{aligned}
$$

Because optimizing this objective function involves simultaneously solving for thousands of variables, we adopt a multi-stage approach to reconstructing the 3D points. First, the points are reconstructed without any strain information because each 3D location can be computed independently. Because many observational errors occur at the scale of the large triangles, we minimize a coarse scale version of the global objective function to produce a start-point for the final optimization problem.

Even with a good starting point, this large optimization problem is intractable without careful attention to detail. First, we reduce computation in numerically computing the gradient by exploiting conditional independence between points on the surface that are significantly far apart. Second, by exploiting the connectivity structure of the surface, we constrain numerical estimates of the Hessian to a sparse matrix form (c.f. [Triggs et al 00]).

The combined strain reduction, point reconstruction creates reconstruction results that are significantly better, yet has little effect on the reprojection errors: typically an increase of less than a fraction of a pixel (in figure 7.9, this reprojection error increase is 0.03 pixels) Because the most accurate views of a triangle are typically separated by a

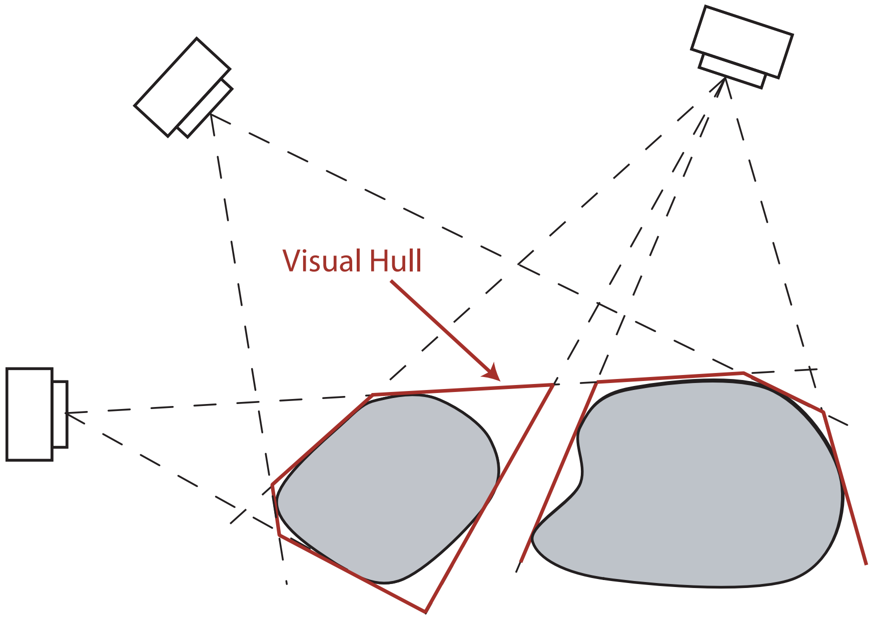


Figure 7.11: Visual hulls can be computed from silhouette edges in images and provide volume constraints on the resulting cloth reconstruction. The visual hull is an outer bound on the location of the cloth.

small baseline, small errors in localization become large errors in depth estimation. The strain reduction only needs to have small effects on the reprojected location of the point to dramatically increase the quality of the reconstructed mesh, as shown in Figure 7.10.

7.5 Using Silhouettes

We use silhouettes to build volume constraints that confine reconstruction for points that are only viewed in a single camera. When combined with strain, the reconstruction of these points is still not as accurate as multi-view reconstructions, but the errors are typically small with respect to the scale of the cloth (figure 7.12).

We motivate this section with the observation that real clothing is difficult to view in its entirety — typically an arm or a leg gets in the way. As figure 7.13 shows, it not

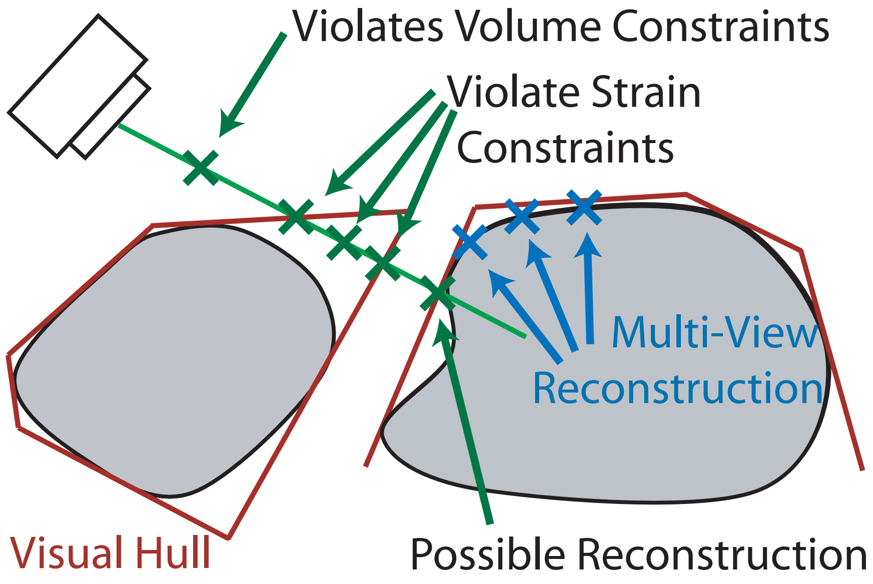


Figure 7.12: 3D locations can be computed from a single view of the cloth by combining other cues. Using the visual hull and the ray extending from the camera, the range of possible reconstructions is limited to several line segments. Because the amount of strain in the cloth is limited, the number of choices is cut down even more. The final 3D point is chosen to be the closest point to the camera that satisfies all constraints.

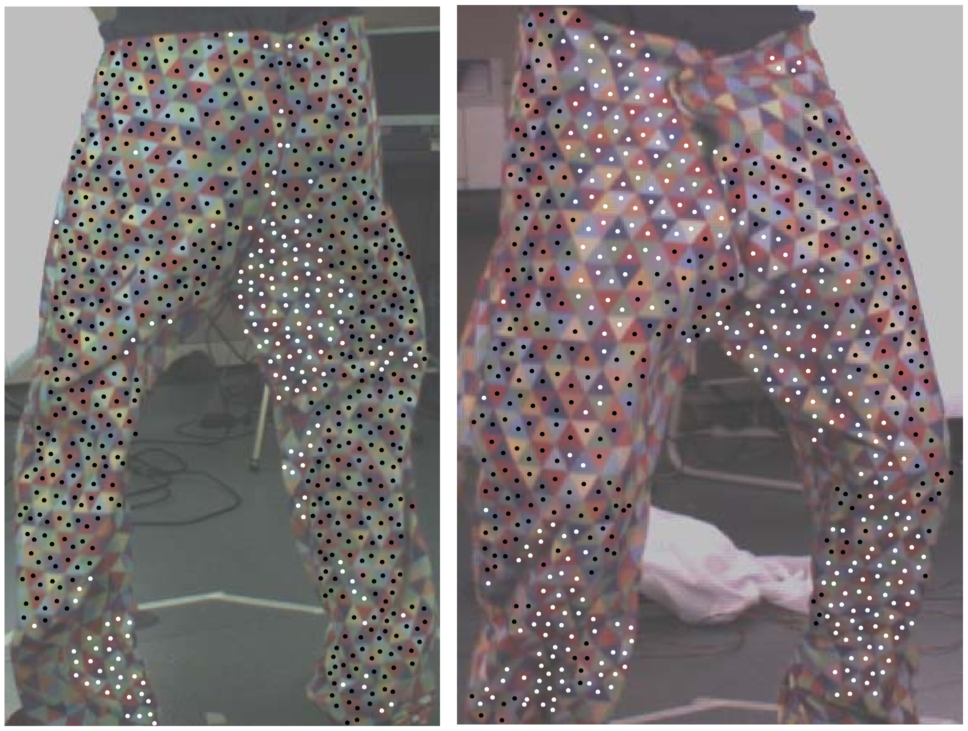


Figure 7.13: With only six cameras, it is common for points to be seen in only one view. We can still use these points for reconstruction by confining the point to a ray and using volume and strain constraints (section 7.5). In the images above, **black points** are detected in at least two cameras while **white points** are viewed in only one camera. Notice that the colors in these two images are significantly different: they were taken using the same model cameras with identical settings at the same moment in time, yet the internal processing of the cameras yields different colors.

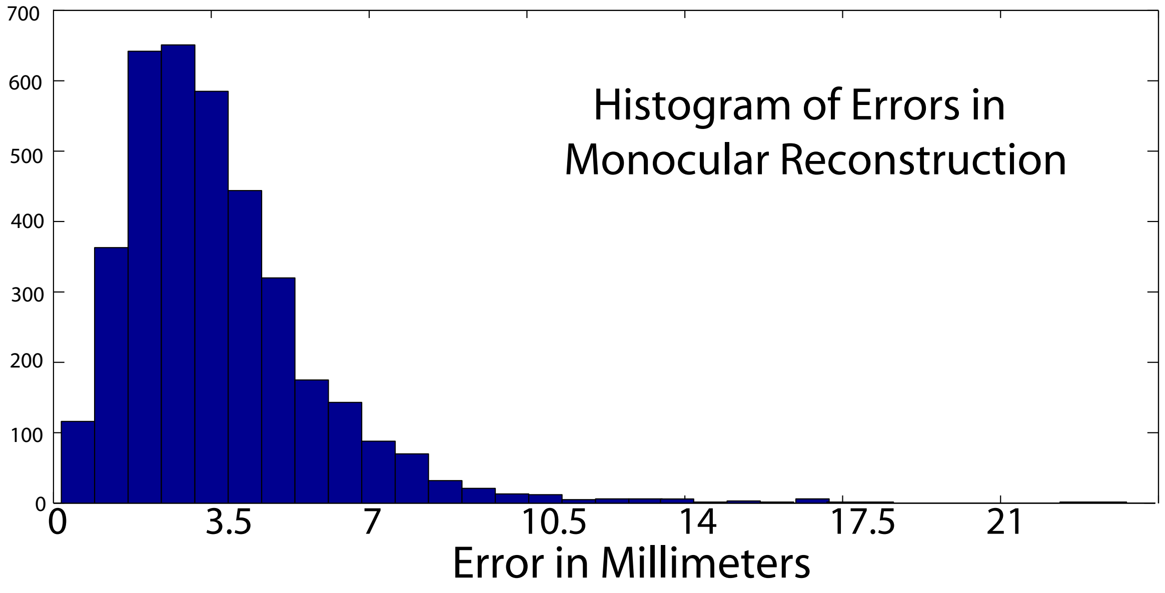


Figure 7.14: Many points must be reconstructed from only one view. Our method does this using a combination of cues, described in section 7.5. We evaluate the quality when reconstructing these points by running the same method on points seen in more than one view and measure the distance between the two reconstructions. The histogram of distances above shows that these errors are still small with respect to the 1m+ scale of the overall object. The median error is 3.3 mm or roughly 0.3%. For comparison, the median reprojection error for mutli-view reconstructions is 0.9 mm.

uncommon for a portion of the cloth to be visible in a small range of views. As a result, using conventional structure from motion, many of these features will either be discarded (if only seen in one camera) or poorly reconstructed due to a narrow baseline (if seen by two cameras that are close together).

To capture silhouette constraints, we start with a grid of voxels covering a volume that is slightly larger than the range of points reconstructed using multi-view techniques (this is reasonable, since exterior points tend to be easily recognized). Using a segmentation of each image into foreground and background, we remove all voxels that project onto the background of each image. The resulting object, known as the visual hull, is a rough approximation of the shape of the object (see figure 7.11 for a schematic example).

Points viewed in a single image are constrained to a ray in 3D that extends from the focal point of the camera through the observation of the point on the image plane into the rest of the scene. We reconstruct these points by projecting the point onto the closest surface of the volume that obeys strain constraints (figure 7.12). Because the visual hull is a coarse approximation to the geometry, it isn't uncommon that the closest surface to the camera is actually on a different portion of the object. In most cases, strain constraints remove this possibility. Then, just like points reconstructed using multi-view techniques, points viewed in only one camera are fed into a combined reprojection error strain reduction optimization. In this case, each point has reprojection error in only one image.

Our representation of the visual hull is common to other reconstruction techniques and could be expanded. For example, we carved out voxels of the scene that appeared between the cameras and their multi-view reconstructions, but found that in general this had little effect on the monocular reconstruction. In the future, we plan to run space carving on the remaining voxels to obtain better surfaces for projecting the points.

7.5.1 Background Subtraction

In order to extract silhouettes, we need to perform background subtraction. The most challenging aspect of this problem in this context is shadows: the shadows are largely lit by reflected light and thus the same object in shadow has a different color. So far, we have assumed a somewhat laborious view of cloth capture: a special studio needs to be set up and a custom garment needs to be sewn together. Here, we adopt an approach which requires human input: we require a rough segmentation for a handful of frames from each camera. In practice, this take ten or fifteen minutes to create and gives us a reasonable model for each pixel. Results can be found in figure 7.15.

For each pixel location in each view, we build a model of the color and variation of each pixel after discarding intensity (to give some robustness to variations in lighting). Then, for new images, we classify the color of the pixel: does it fit within our model? (aka is it background?) or is it outside the model? (is it foreground?) When this is done we fill in the holes and select the largest foreground object in view. We assume that this object is the cloth object and that all other pixels are not on the cloth object.

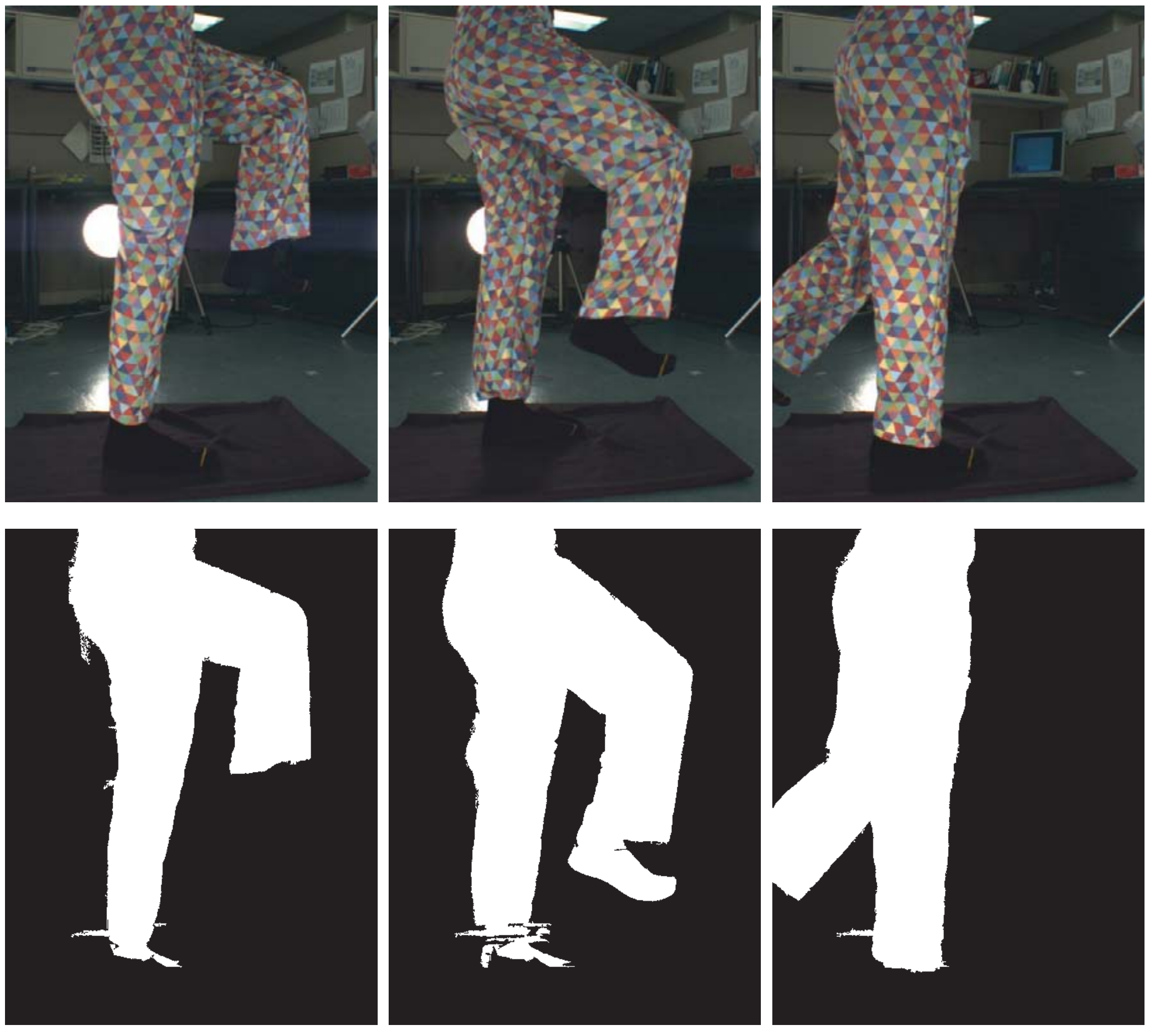


Figure 7.15: Some results for our background subtraction algorithm.

Part III

3D Modeling

Chapter 8

Modelling with Examples

This chapter focuses on a number of mesh processing tasks where examples are deformed to produce new meshes. This is particularly useful in two different cases: (1) filling in holes in capture data and (2) creating new animations in a purely data driven way by deforming and combining examples. This chapter has three parts: background on the basic mesh representation that we use (deformation gradients), the application of this representation to hole filling and finally the application of this representation to data driven animation.

8.1 Background

Deformation Gradients Our basic mesh editing tool is the deformation gradient [Sumner and Popovic 04, Sumner et al 05]. The advantage of the deformation gradient is that it allows us to maintain many important aspects of the cloth structure when transferring from one mesh to another. The major features of deformation gradients include: linear

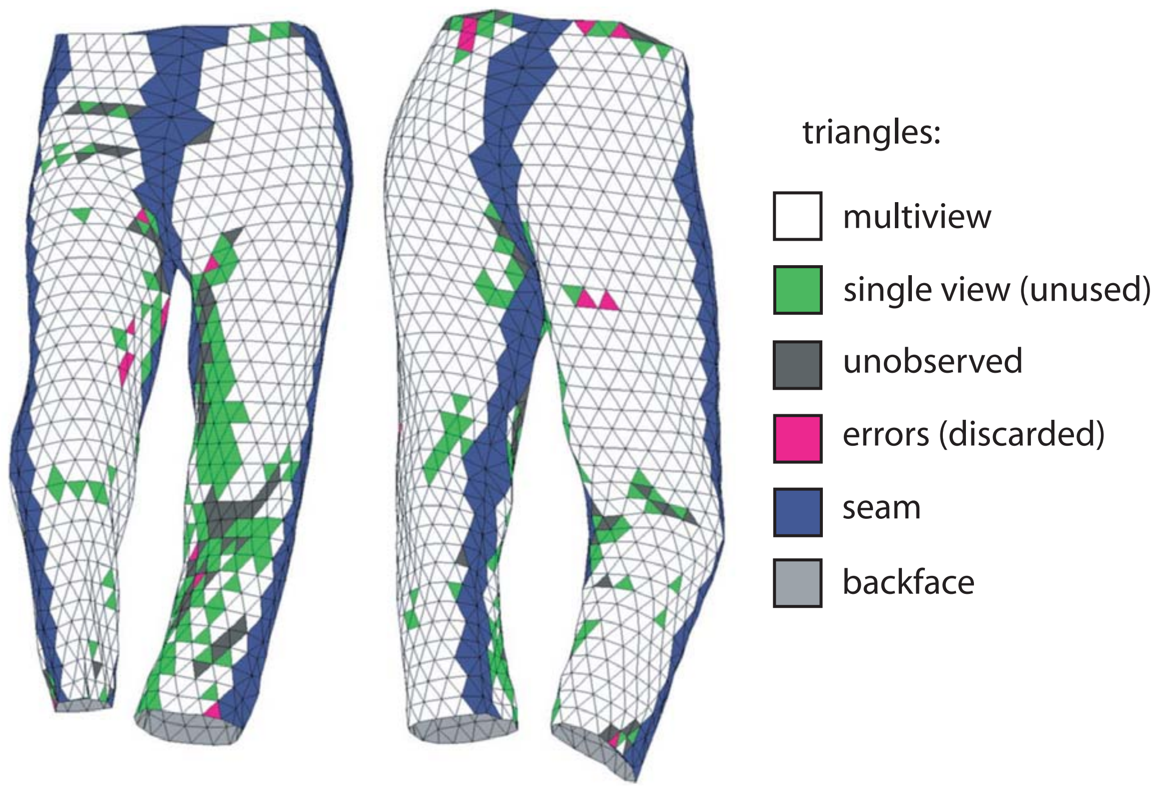


Figure 8.1: Occlusion is inevitable when capturing highly articulated objects. In this reconstruction, the inner thigh region of the left leg is difficult to observe because the right leg shields it from view. Regions that contain errors or that are seen in two or fewer views must be filled afterwards. Errors are detected using reprojection error and strain.

reconstruction of point locations from deformation gradients, the ability to specify hard constraints on point locations, and the intrinsic penalization of large changes in scale or skew – both of which are unlikely in cloth.

Formally, the deformation gradient is the Jacobian of the affine mapping between the points in the source mesh and the target mesh. Typically, this transformation is defined for triangles, however, triangles do not define a complete mapping. As a result, for each triangle, a forth vertex needs to be defined that is out of the plane of the other three. Following the basic convention in [Sumner and Popovic 04], we add a forth vertex in the normal direction (note: we move the forth vertex away from the mean of the points instead of the first vertex):

$$v_4 \quad = \quad \frac{v_1 + v_2 + v_3}{3} + \frac{(v_2 - v_1) \times (v_3 - v_1)}{\sqrt{\|(v_2 - v_1) \times (v_3 - v_1)\|}} \tag{8.1}$$

Now, using v to represent the source triangle and $\tilde{v}$ to represent the target triangle, the deformation gradient is (notation taken from [Sumner et al 05]):

$$T \quad = \quad [v_1 - v_4 \quad v_2 - v_4 \quad v_3 - v_4]^{-1} [\tilde{v}_1 - \tilde{v}_4 \quad \tilde{v}_2 - \tilde{v}_4 \quad \tilde{v}_3 - \tilde{v}_4] \tag{8.2}$$

The key aspect of this equation is that the deformation gradient is linear in the vertex locations $\tilde{v}$. As a result, if we know the deformation gradients, we can reconstruct the point locations by inverting the system. The key to making this successful is to build up a linear system such that all of the deformation gradients are defined in terms of the same set of points. If no points are duplicated, then the solution will fit the deformation gradients in the least squares sense. Following Sumner's notation, we use G to represent the matrix that converts a vector of point coordinates x into a vector of deformation gradients f: $f = G \cdot x$. To solve for point coordinates when deformation gradients are known, we perform the minimization: $x = \arg\min_x \|f - G \cdot x\|$. This is a simple least squares minimization and can be solved using the corresponding set of normal equations.

To specify some point constraints, we define a vector of deformation gradient constants based on the locations of the points that we know. This vector of constants is $c = G \cdot x_0$ where x_0 is the vector of point constraints with zeros for all unknown points. To minimize using least squares, we remove the rows of G and x that are constrained and proceed as described above.

MeshIK: [Sumner et al 05] consider the problem of combining multiple examples to generate an output mesh with specific vertex constraints. The basic technique is to search for weights that average deformation gradients to produce the minimum residual error. In the case that the combination is a linear weighted sum, this process is again inversion of a linear system. In this case, we use M to represent a collection of deformation gradient vectors (previously f) organized by column. A weighted sum of deformation gradients becomes a matrix vector product:

$$x, w \quad = \quad \arg\min_{x,w} \|M \cdot w + c - G \cdot x\|$$

However, linearly combining deformation gradients defies their semantics: each deformation gradient has a rotation component and a stretch / skew component. Performing a linear weight average of rotation matrices can cause undesirable output. Instead, [Sumner et al 05] do a polar decomposition of each deformation gradient into the rotation and stretch / skew components. The stretch skew matrices are combined linearly, but rotation matrices are combined multiplicatively (where the weights are the exponents). Computationally, this is performed using the matrix logarithm and the matrix exponential. The corresponding system is nonlinear:

$$x, w \quad = \quad \arg\min_{x,w} \|M(w) + c - G \cdot x\|$$

As a result, [Sumner et al 05] use a nonlinear minimization method to solve for the weights w and the points x. This method works by repeatedly linearizing the problem and solving for points, weights and updates.

Editing Mesh Sequences: [Kircher and Garland 06] define a number of tools

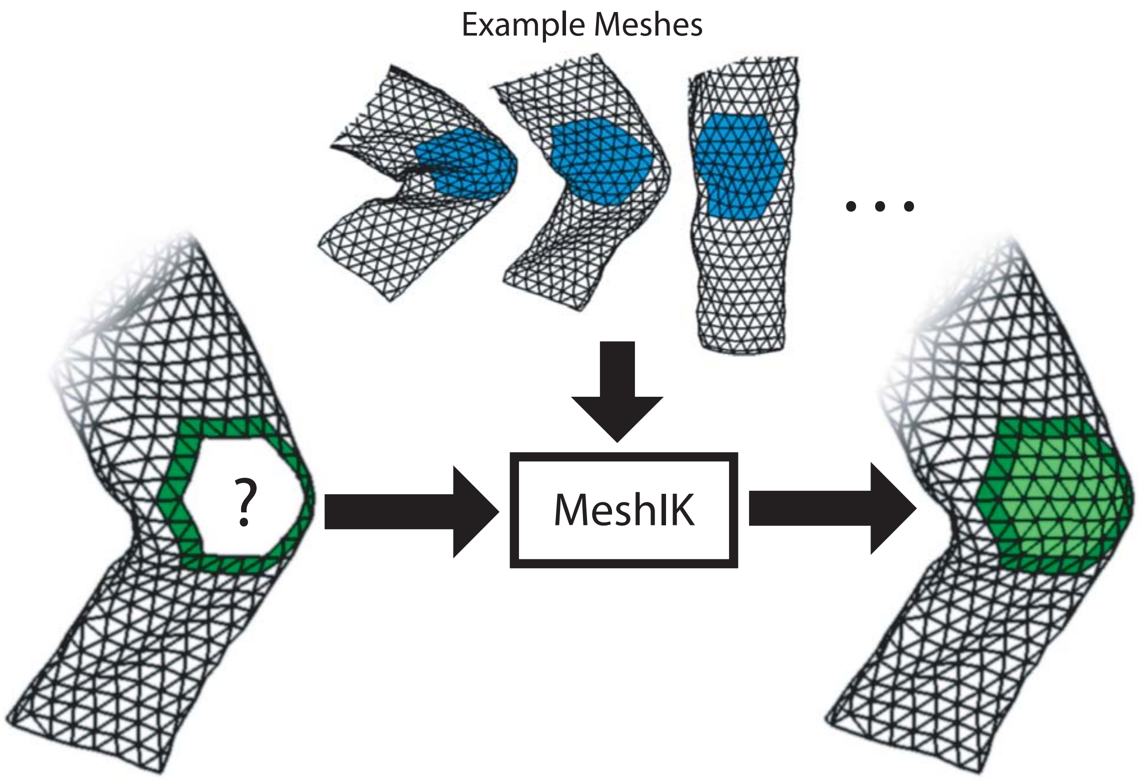


Figure 8.2: Holes are filled with a combination of cloth sections observed in other frames. (In reality, we use a ring of two triangles as constraints)

to edit mesh sequences. These tools are particularly useful for our application: they allow us to modify the geometry after capturing to suit a new purpose. There tool is particularly suited to editing the geometry: they use a multiresolution framework to propagate edits to other frames in the sequence. We don't adopt their internal representation, but use their tools to modify our data to get new effects.

8.2 Hole Filling

In the acquisition process, occlusion inevitably creates holes in the reconstructed mesh (figure 8.1). One would like to fill these holes with real cloth. One of our major contributions is a data driven approach to hole filling: we fill holes with previously observed sections of cloth.

This hole filling procedure has a number of requirements: the missing section needs to be replaced by a section with the same topology; the new section needs to obey a number of point constraints around the edge of the hole, and the splicing method should respect properties of cloth (specifically strain). We select a reconstruction technique based on deformation gradients as the method of choice [Sumner et al 05]. In this approach, we fit deformation gradients for the missing section to a combination of deformation gradients in other observed sections. Then, we reconstruct the point locations from the deformation gradients.

This procedure has a number of advantages. First, deformation gradients naturally yield cloth like properties. Deformation gradients are the transformation matrix between triangles in two poses of the mesh. By penalizing elements that deviate in this matrix, we have a fairly direct penalty on large changes in scale or strain. In contrast, methods based on the Laplacian of the mesh ([Sorkine et al 04]) do little to penalize these strains and can show many artifacts around the edge of the mesh. Second, deformation gradients can be converted into vertex locations by inverting a linear system, allowing us to specify vertex locations as constraints. Methods such as [Lipman et al 05] don't allow vertex constraints.

8.2.1 Implementation

We use occlusion free meshes from other frames to automatically interpolate holes. For each hole in each frame, we cut out the missing region plus a ring of two triangles

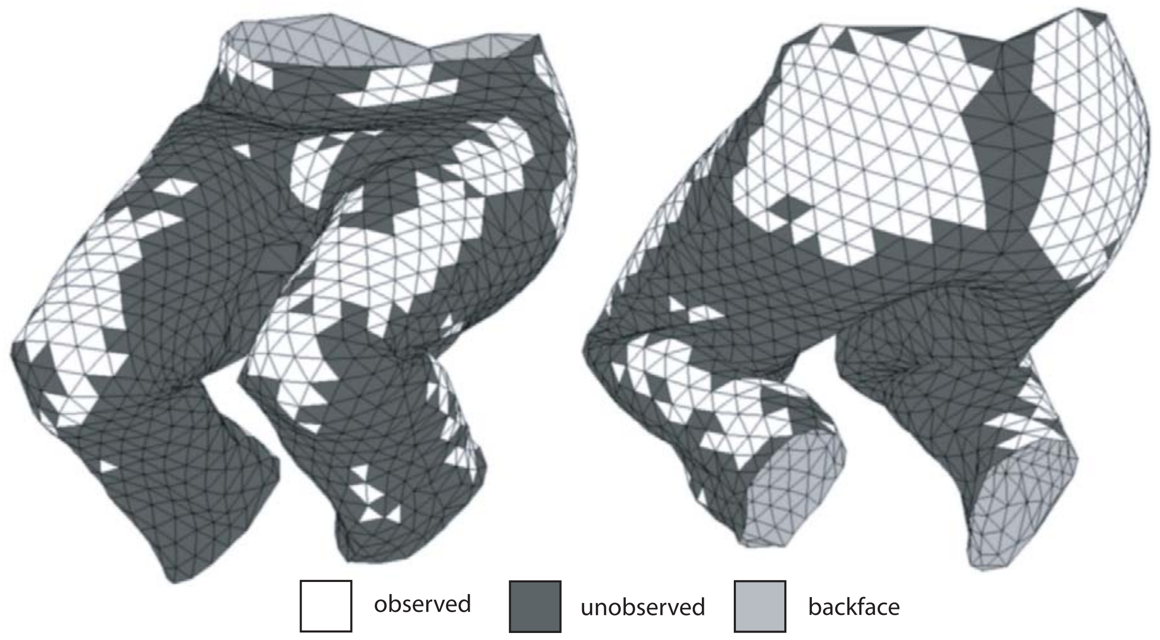


Figure 8.3: Our hole filling works in extreme circumstances. In this frame, 73% of the mesh is unobserved and inserted using MeshIK based hole filling. This frame is unusual: only 22% of the surface is unobserved in an average frame.

around the region. We select a set of examples of the enlarged region, then use MeshIK ([Sumner et al 05]) to reconstruct the surface. MeshIK works by choosing a combination of deformation gradients from the examples and then solving for the missing point locations. We use the points from the ring of known triangles around the hole as constraints in MeshIK.

The most restrictive aspect of MeshIK is that it requires example meshes without holes. In practice, we *never* observe complete example meshes – each mesh is missing some triangles. These holes appear in different places in different meshes and we create complete meshes in an iterative method. First, we fill all holes with a naive linear algorithm (specifically, we triangulate across gaps and use barycentric coordinates to place the missing points – this gets the job done, but works poorly). Then, we do another pass through all

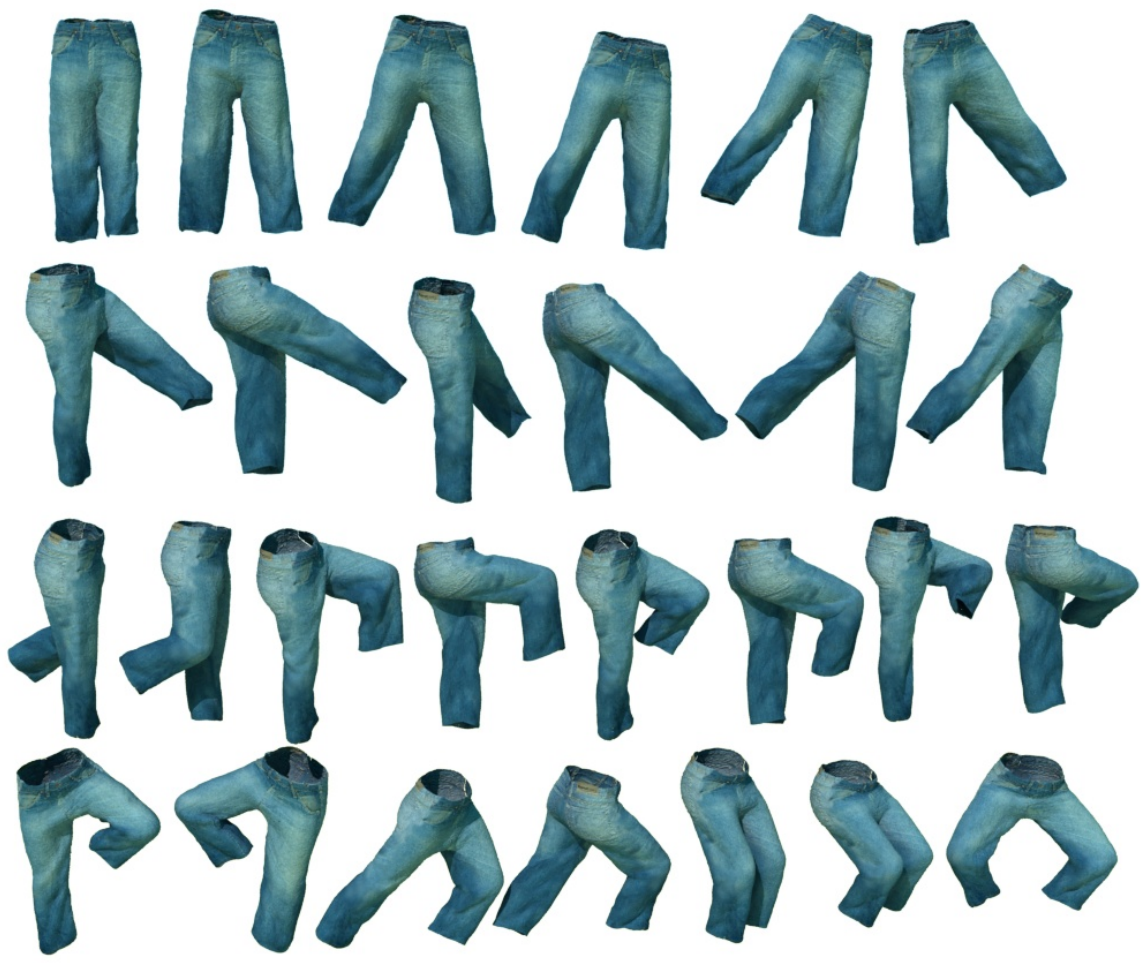


Figure 8.4: This figure shows the twenty seven basis poses that we collected to capture major modes of deformation in the pants. These poses were used in creating the motion capture sequence and for hole filling in the captured sequences.

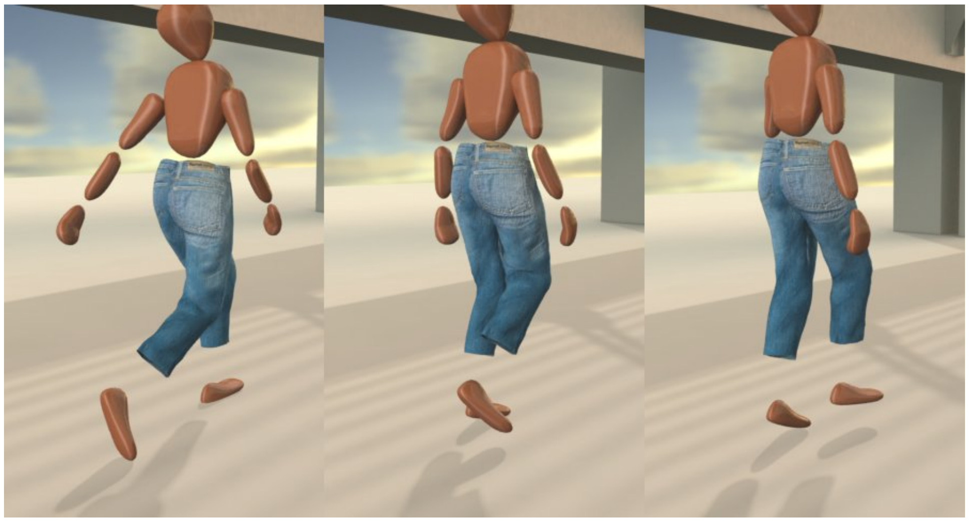


Figure 8.5: Here, we use human motion capture data (the skeleton) to animate cloth by using MeshIK on a frame by frame basis.

the data, where we replace the linear sections with sections created using MeshIK on the linearly filled data. To down weight the linear data, we select the examples with the highest percentage of viewed points in the missing section. These frames are then used as examples in MeshIK to hole fill in the rest of the sequence.

For the pants capture, we iteratively refine a set of 27 extreme poses which were captured specifically for filling holes. The advantage of this approach is that the example poses are chosen to capture the relevant degrees of freedom – yielding better results. For the cloth toss sequence, we chose the simpler approach: iteratively refined the entire sequence.

8.3 Adding a Skeleton to Cloth Capture Data

In order to animate cloth on human motion capture data, we need a correspondence between our cloth capture data and the skeleton in the human motion capture data. We use the 27 basis poses from before (figure 8.4). For each of the 27 poses, we inserted joints

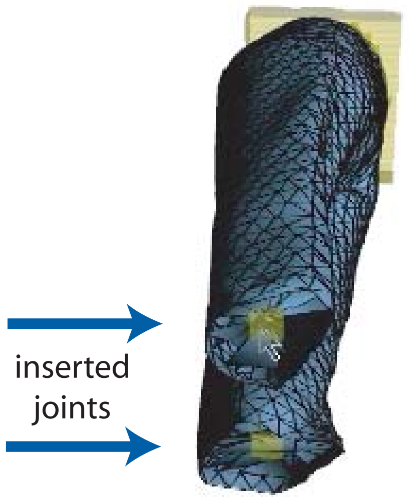


Figure 8.6: Joints are added to meshes to fit to mocap data. Specifically, for the pants model, we insert six joints: two at the hips, two at the knees and two at the end of the pant legs (between the knees and the feet).

by hand (figure 8.6). These points are then connected to the mesh by inserting roughly a dozen triangles that connect each joint to the nearby mesh. Now, we can specify joint locations and examples and use MeshIK to reconstruct the cloth surface.

Later in this chapter, we will need skeleton joints for the entirety of our cloth capture data – which is currently more than 2000 frames. In this case, insertion by hand is both impractical and error-prone. Instead, we insert these joints automatically using the 27 basis poses. For each frame in the database, we use all of the points on the surface of the cloth as constraints and treat the six joint locations as unknowns. Then we use MeshIK with the 27 basis poses to reconstruct the location of the six joints.

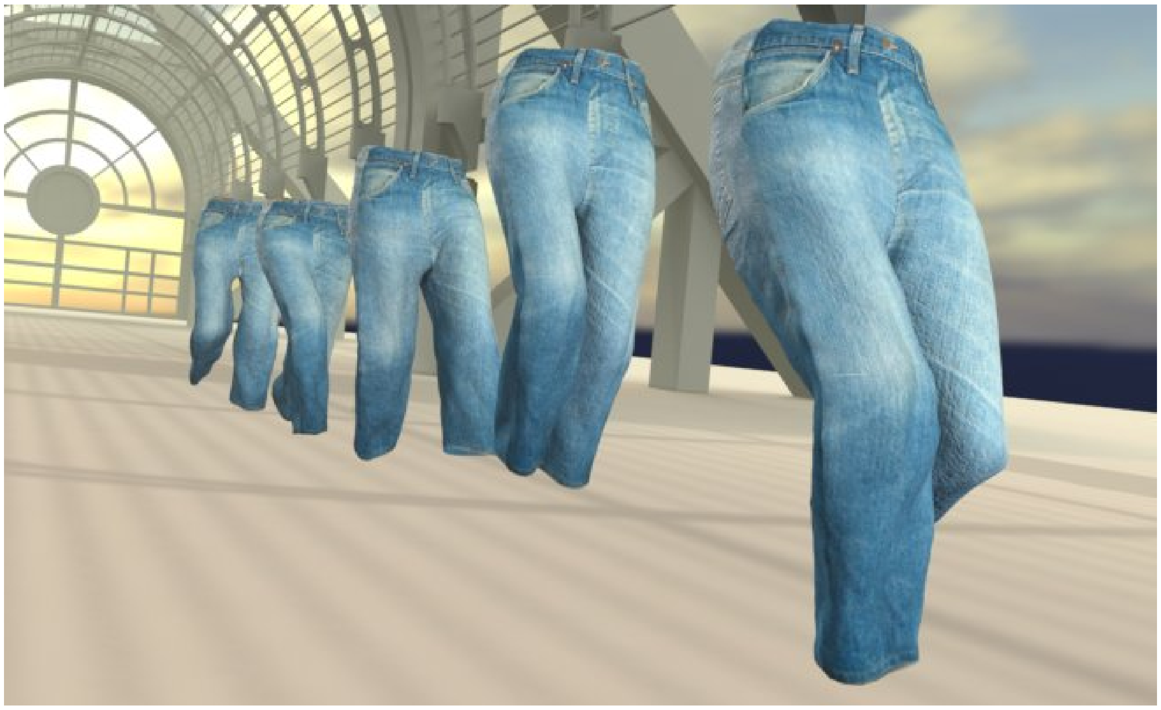


Figure 8.7: We use MeshIK to bind captured cloth to human motion capture data using 6 joints in the mocap data.

8.4 Animation using Kinematics

We use a small set of captured frames (the previous basis of the 27 examples) in combination with MeshIK to skin skeletal human motion capture data (figure 8.7). This approach covers a reasonably large range of motion, but ignores cloth dynamics.

Here, we use the inserted proxy points for knee and hip joints from each of our basis meshes. We use the average bone lengths in the example meshes for our new skeleton. We use the joint angles from the human motion capture data. The resulting joint locations are used as constraint points in MeshIK, which produces the final output meshes. Using our MATLAB implementation of MeshIK, this process takes around 5-10 seconds per frame.

In order for a small basis to adequately express a full range of motion, each basis pose must be an *extreme* configuration. For simple objects such as a cylinder, a small bend (for example) is sufficient to extrapolate to a larger bend [Sumner et al 05]. However, for pants the relationship is more complex: the fact that no folding occurs in a small bend does not imply that folding will be absent in a larger bend. Conversely, if a decent amount of folding occurs in a small bend, we do not expect extreme folds in a corresponding larger bend. As a result, MeshIK is most useful when a basis is carefully chosen to prevent extrapolation artifacts.

One drawback to our approach is the loss of secondary kinematic motion, such as the sway of loose cloth. Because MeshIK does not use velocity information, the resulting animation appears damped.

8.5 Animation Using Dynamics

In this section, we animate cloth by example by splicing together short sequences of existing cloth animation. In contrast to the previous section, this approach models significant dynamic effects such as the secondary motion of the cloth on faster leg movements.

In general, we try to avoid smoothing whenever possible. Spatial smoothing removes folds and wrinkles – characteristics of cloth the we wish to preserve. Smoothing in time removes secondary motion in the cloth – the motion induced in the cloth by the movement of the skeleton.

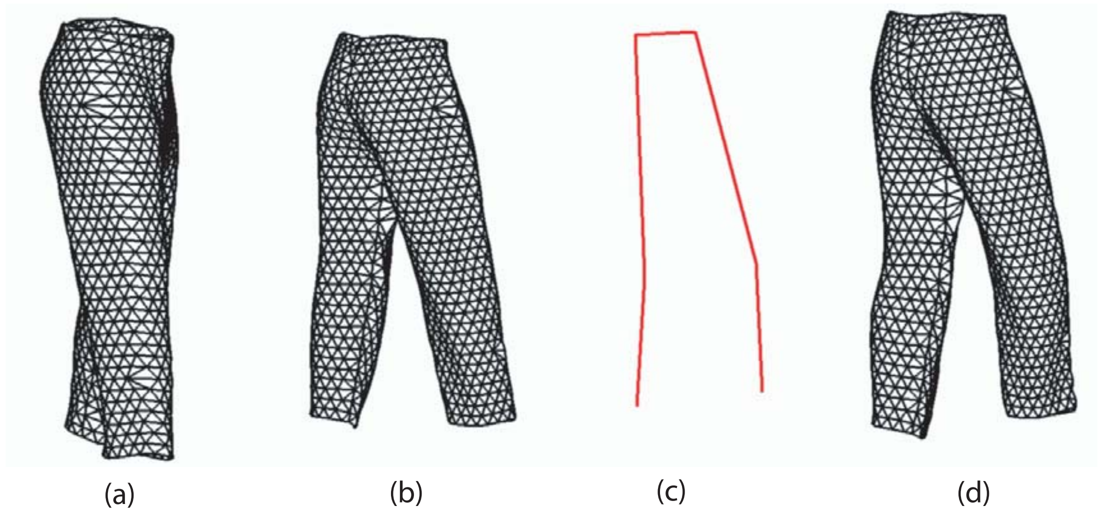


Figure 8.8: An example from the database can be deformed to fit a target example. **(a)** An example pose from the database. **(b)** the same example pose, rotated to match the orientation of the target skeleton. We only allow rotations around the vertical axis to preserve the direction of gravity. **(c)** The target skeleton. **(d)** the database pose warped to fit the skeleton. We transfer the deformation gradients from the surface of the rotated database pose and use the skeleton as constraints.

8.5.1 Deforming the Cloth

The most common operation that we use is a coarse scale deformation of cloth from the database to match a specific pose in the human motion capture. Our underlying representation is the deformation gradient [Sumner and Popovic 04], which is the transformation matrix between triangles in different configurations. In our case, we define deformation gradients with respect to a standard equilateral triangle. To reconstruct the cloth in a new position, we specify the location of a small number of points (the joints in the skeleton) and reconstruct the mesh (linearly) from the deformation gradients. The advantage to this approach is that the joint locations are guaranteed to be correct.

This approach has several benefits over the MeshIK technique put forward by

[Sumner et al 05]: first, it is significantly faster because the optimization is linear (MeshIK is nonlinear); second, because there is only one example of the mesh, there is little smoothing when reconstructing using this method. This is advantageous because it means that the original folds are transferred to the target mesh.

Figure 8.8 shows the process of transferring a mesh. The process involves first lining up the mesh with the skeleton and then transferring the detail using deformation gradients. Directions and alignments are important: MeshIK does not account for model rotation. We allow any mesh to be rotated around the vertical axis, preserving the direction of gravity but substantially enlarging the size of our database by ignoring the direction that the skeleton is facing. In practice, we compute this alignment by aligning the hips alone. We also tried alignment using all 6 of the joints, but found that this was less successful in practice (especially when the points are close to co-planar).

8.5.2 Finding a Sequence

Our final animation is a series of short segments of cloth motion from the database. We find the optimal set of sequences by dynamic programming. Our trellis has two costs: a unary cost between a frame of human motion capture and the cloth database, and a binary cost between frames of the database. These two costs are computed the same way: by combining the distance between the skeleton joint positions and the difference in joint velocities (see figure 8.9 for how these costs are computed). The rest of this section has two parts: defining the generic matching costs, and solving for the optimal path using dynamic programming.

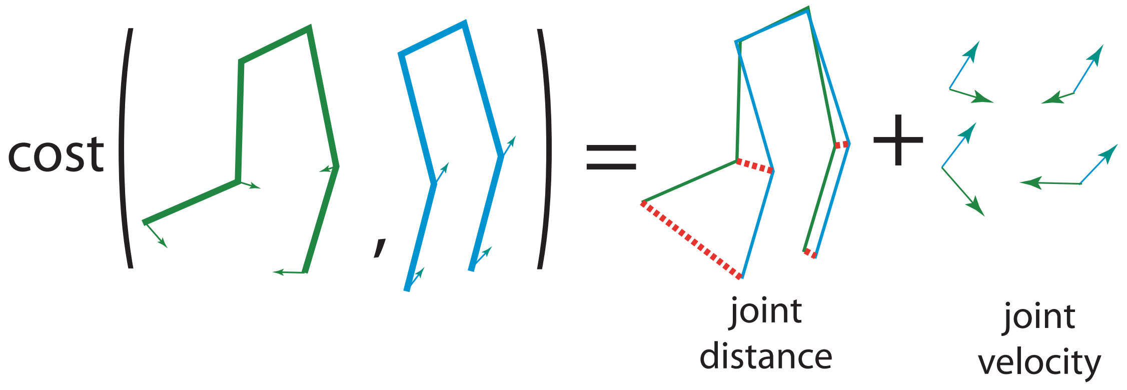


Figure 8.9: We use a cost function with two components to match skeletons. The first component is a similarity in joint positions. The second component is a similarity in joint velocities (computed in the coordinate frame of the parent bone). Figure 8.11 shows the computation of this cost on our database and human motion capture skeletons.

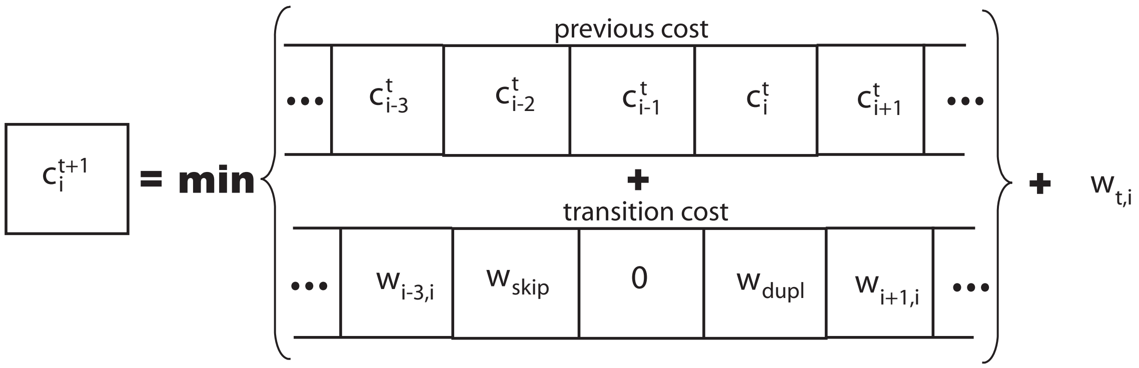

c_i^{t+1} = min
previous cost
... c_{i-3}^t c_{i-2}^t c_{i-1}^t c_i^t c_{i+1}^t ...
+
transition cost
... $w_{i-3,i}$ w_{skip} 0 w_{dupl} $w_{i+1,i}$ ...
+ $w_{t,i}$

Figure 8.10: To find a path through our database, we use dynamic programming with incentives to find a contiguous path through the database. c_i^t is the minimum cumulative cost at frame t using mesh i in the database for the mocap skeleton at time t. $w_{i,j}$ is the cost to transition from frame i in the database to frame j in the database: see figure 8.9.

Matching Cost When we transfer the motion of the cloth from the database to the motion capture database, we must be careful of two aspects: (1) we don't want the cloth deformation to be too significant and (2) we would like the motion of the cloth to be realistic. As a result, we define a matching cost that is the sum of two different terms: the distance between the joints in the skeletons and the distance between the joint velocities.

The process for computing a matching cost is entirely based on the skeleton. This procedure has the advantage that its works for any combination of our data: combining the database of cloth capture with the human motion capture skeletons. The disadvantage is that dynamic properties of the cloth aren't covered directly: only the dynamic properties of the skeleton are used in the matching cost. If one had a very large database, it would probably be worthwhile to define a new matching cost to take into account the dynamics of the cloth itself.

Our matching cost procedure consists of several parts: rotate the skeletons, compute the distance between corresponding joints, compute the joint velocities, compute the cost between corresponding joint velocities. The final score is the sum of the distances and the joint velocity costs.

In the first stage, we rotate the skeletons around the z-axis so that the hip joints line up. As noted before, aligning the full set of joints is less successful in practice. Next, we compute the l_2 distance between joints. This part directly penalizes differing configurations of the skeletons. After that, we compute the joint velocities. We compute joint velocities in reference to the thigh. The goal is to remove the effect of joint position (since it was already taken into account in the previous stage). Specifically, the left knee and the left

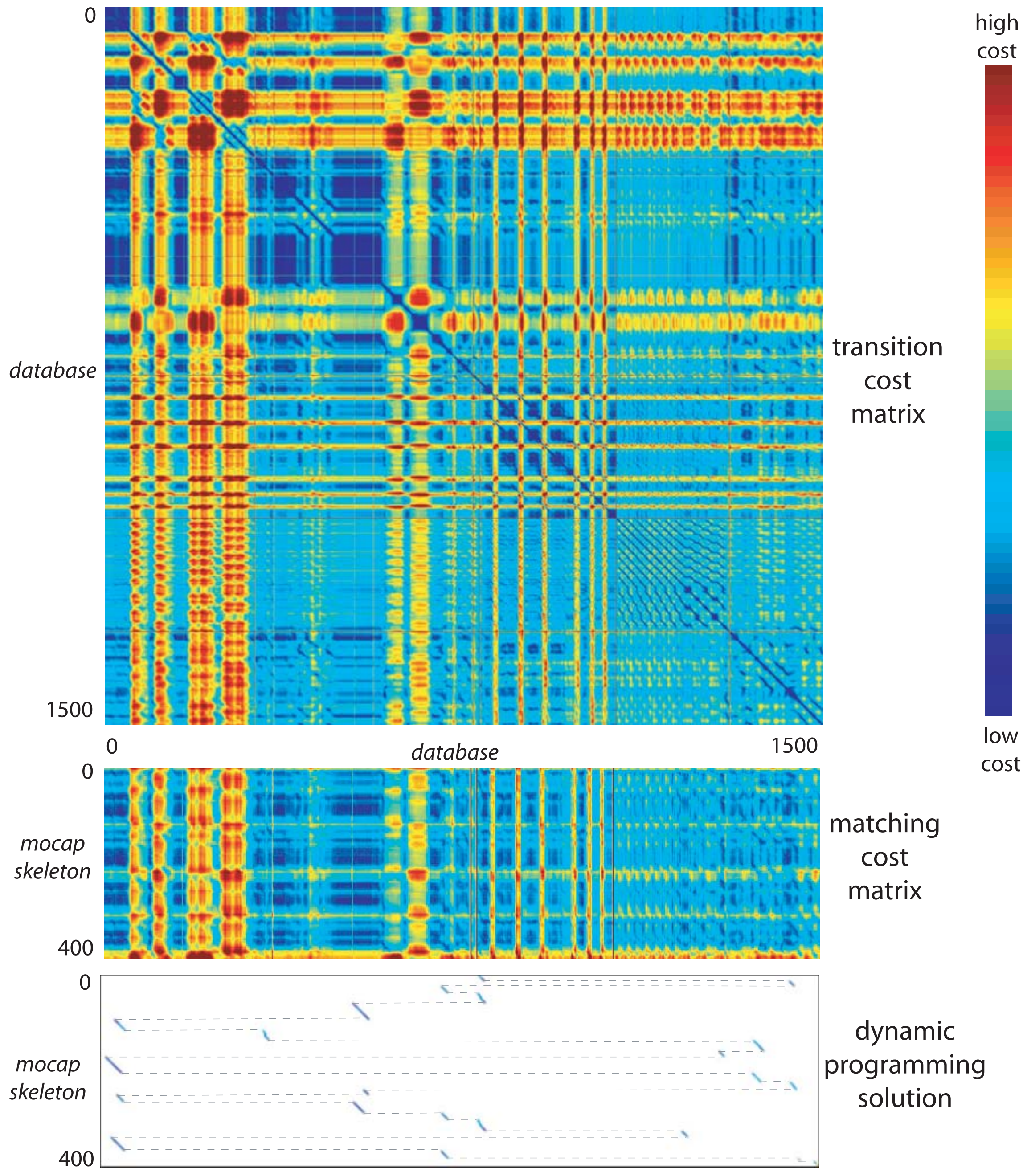


Figure 8.11: These two matrices respectively show the matching costs between the database and itself and between the database and skeletal motion capture data. Diagonal bands in the matrices reveal sequences of self similarity – such as regions where the mesh is repeating a similar motion over and over again. The bottom image is a solution through the database: diagonal structures indicate matches that occur over several frames (see figure 8.10 for the modifications to the transition cost to promote longer sequence matches).

foot velocities are computed in the reference frame of the left thigh. Likewise, the right knee and the right foot velocities are computed in the reference frame of the right thigh.

We compute the joint velocity costs as the norm of the difference in the joint velocity vectors. Finally, we combine the two costs as a weighted sum of the position and velocity costs. To get roughly equal contribution from the two terms, we set the weight on the position cost to be one third of the weight on the velocity cost.

This process is highlighted in figure 8.9 and examples of the resulting cost matrices are in figure 8.11.

Dynamic Programming To find the optimal sequence of database frames to use in a new animation, we use dynamic programming. Our state space is the size of the database: for every frame in the target sequence, we assign one and only one frame from the database.

To find this path, we consider two different costs. The first cost is the cost to match a frame in the database to a frame of mocap. This cost is the cost defined in the previous paragraphs and consists of two parts: the position cost and the velocity cost. The goal is to pick frames that are similar and map them onto the target.

The problem becomes more difficult when we consider dynamics: visually, the results look better when subsequent frames in the new animation are subsequent frames in the database. As such, we define a transition cost such that there is no penalty for a transition that preserves the sequence match.

However, it is impossible to map a single sequence in the database to an entire mocap sequence: transitions are necessary. In addition, not all transitions are the same.

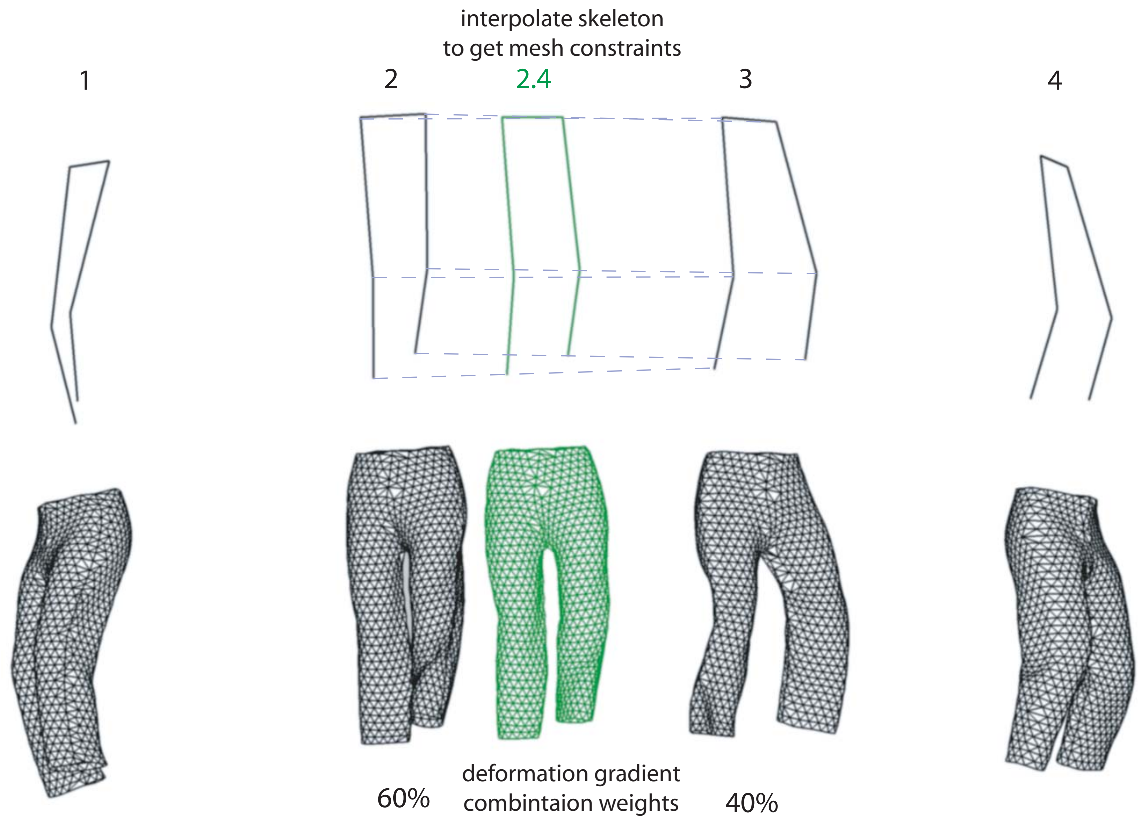


Figure 8.12: In order to resample in time, we need to interpolate mesh configurations. We do this by linearly interpolating the skeleton and use the joints as constraints. We reconstruct the mesh surface by doing a combination of deformation gradients.

Transitions between similar frames in the database are more desirable than transitions between frames in the database that are dissimilar. As a result, we use the matching cost between skeletons in the database as the transition cost in dynamic programming. Note that we have already defined the transition to the next frame as zero cost.

Finally, we have to worry about time alignment. Minor misalignments in the timing of a sequence shouldn't force transitions or large costs. To take this into account, we use a two stage strategy. First, we make transitions that duplicate or skip frames in the database lower cost. Second, we include a large double duplication penalty (technically this is a non-Markov property, but it can still be added to dynamic programming easily). In the next section, we will define a subsampling operator that allows us to use this time alignment system without causing significant visual artifacts.

After running dynamic programming on several sequences, we get paths through the database. A path (visualized in the bottom of figure 8.11) is typically composed of blocks where there is a rough alignment between the timing of the database and a timing of the mocap sequence. In practice, these blocks are typically 20 frames in 24 fps data.

8.5.3 Resampling in Time

Because our dynamic programming allows for duplication and skipping of frames, we need to resample the database meshes in time to produce smooth animations. For example, if the motion in the database is half the speed of the mocap motion, we would see a path like 1,1,2,2,3,3. A faster sequence might be 1,3,4,6. Unfortunately, duplication or skipping of frames can be quite noticeable.

We define a subsample operator that allows us to interpolate frames and use fractional indices. In the examples given above, we may sample at 1,1.5,2,2.5,3,3.5 and 1,2.5,4,5.5. This subsample operator works in two parts: first, we interpolate the location of the underlying joints in the skeleton. In practice, linear interpolation is good enough. Second, we compute the mesh deformation gradients by creating an appropriately weighted combination of deformation gradients from the two neighboring frames (using the non-linear deformation gradient combinations defined in [Sumner et al 05]). We then reconstruct the surface of the mesh using the interpolated skeleton as weights and the combined deformation

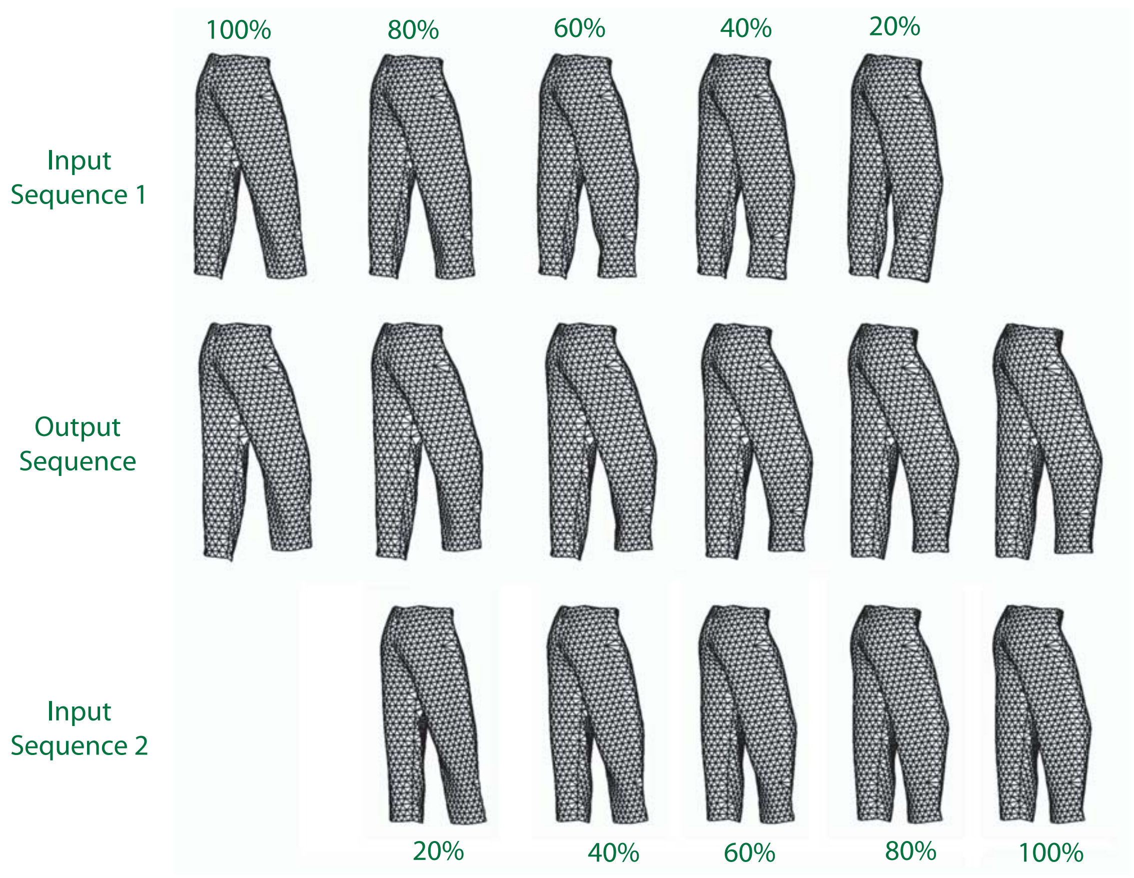


Figure 8.13: To create new animations, we blend together small sections of motion from our database. However, cuts between sequences are harsh, so we blend deformation gradients over several frames for each transition.

gradients. Figure 8.12 shows a graphical version of our mesh interpolation.

However, before we can interpolate frames, we need the fractional values. To compute these fractional values, we smooth the path. Specifically, we compute the numerical derivative of the path, smooth with a $\sigma = 2$ frame Gaussian, then reintegrate to get back the original path.

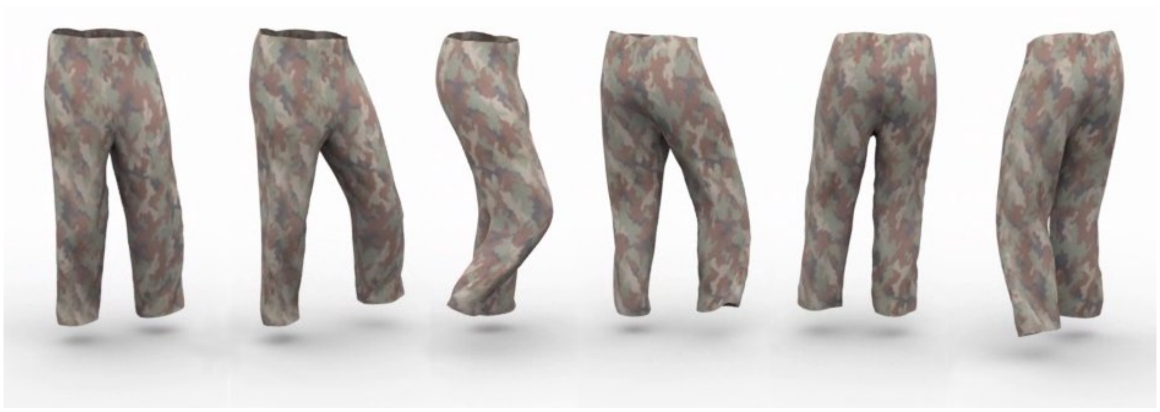


Figure 8.14: Using a database of cloth motion, we tie animation to a skeleton by concatenating and smoothing short clips from the database.

8.5.4 Smoothing Transitions

The cloth reconstructions produced in the previous step have obvious jumps at every transition between frames in the database. Often, these jump occur because small folds in the cloth move, creating visible artifacts. Even with a large database and a more powerful matching cost, this is inevitable: cloth contains significant state information.

To make these jumps less noticeable, we smooth the transitions (figure 8.13). As stated in the introduction to this chapter, smoothing is undesirable – it removes some of the high frequency detail that is so characteristic of cloth. We do this smoothing purely in the deformation gradient domain. Since are reconstructions at this point use the mocap skeleton as constraints, there is no reason to interpolate the skeleton as we did to interpolate meshes.

Instead, we use linear weights on the deformation gradients. Our smoothing time frame is 5 frames. This constant is a balance: a larger constant gives more fluid transitions but has the potential to remove more folds.

8.5.5 Results

We use a database of roughly 1500 frames of captured cloth pants to animate cloth on two different sequences of human motion capture data. Our capture data is lacking any walking motion, and as a result we have some artifacts in animating walking and running motions. However, animation is realistic for other motions. We believe that data-driven cloth animation is a viable alternative to cloth simulation – especially for cases like video games where large amounts of cheap animation is needed.

Chapter 9

Conclusion

9.1 Summary

In summary, we have provided a suite of different tools that can be used for building models of cloth from images and for creating cloth animations from these models. On the modeling building side, we developed tools to build texture maps from individual images, tools to build 3D models from individual images and tools to build 3D models from groups of simultaneous images.

These tools place increasing constraints on the capture of the images. In the first case, we require that the surface be *screen print*, or composed of a small set of distinct colors. However, performance is better when the pattern on the clothing is custom printed. In the next case, building 3D models from individual images, we again have the choice: we can build models from existing screen print textures or we can build models custom patterns. In this case, we place stronger constraints on the lighting: specifically we assume that there is a single infinite point light source. However, our method is robust to small errors: minor interreflections or a moderately nearby lightsource can be tolerated. Finally, when building a 3D model from multiple images, we require that the surface of the cloth have a specific pattern.

While the tools require increasing constraints on the recording of the cloth, they also produce increasingly accurate and flexible models. In the first case, texture replacement, there is little flexibility: the texture can be replaced but the lighting and aspect are immutable. In the second case, 3D from individual images, minor adjustments in aspect and lighting are possible, but major adjustments are likely to produce problems. Finally, with a multiple camera setup and a custom pattern on the cloth, we can build complete 3D models. These models are so powerful that we can not only change texture, aspect and lighting, but we can even edit the underlying motion of the surface.

From an applications perspective, it is important to pick the appropriate tools: adding clothing to a video game character requires the quality and control of the full 3D capture. Meanwhile, logo replacement in media only needs the simpler texture mapping.

9.2 Limitations and Drawbacks

Each of the methods in this paper includes a number of limitations and drawbacks. The following subsections address these according to different applications.

9.2.1 Texture Tracking

For texture replacement and 3D estimation from individual images, we require screen print patterns and either a frontal view of the surface or a video of the surface with substantial motion. The largest downside is that estimation of the surface geometry can't be done from an individual image alone.

Second, these approaches require some sort of existing texture on the surface of the fabric. This is not necessarily true in all cases: single color clothing items are somewhat common. The work of [Fang and Hart 04] address this case by using optical flow to track movement on the surface and by using shape from shading to make rough estimates of the surface orientation for texturing. In addition, existing textures that are highly repetitive present problems as well. [Lobay and Forsyth 04, Lobay and Forsyth 06] address the problem of estimating geometry from repeating textures, [Hays et al 06] address the problem of extracting the structure of this repeating structure and [Liu and Lin 03, Liu et al 04, Lin 05, Lin and Liu 06a] collectively address the problems of tracking and retexturing regular and *near-regular* textures.

Finally, our texture tracking procedure is slow – often tens of seconds per frame. There is a vast literature in this area and we refer readers to [Pilet et al 05] for a faster method and a more complete bibliography. The advantage to our tracking method is that it doesn't require an energy model and thus reports a model that most accurately describes what is in the image.

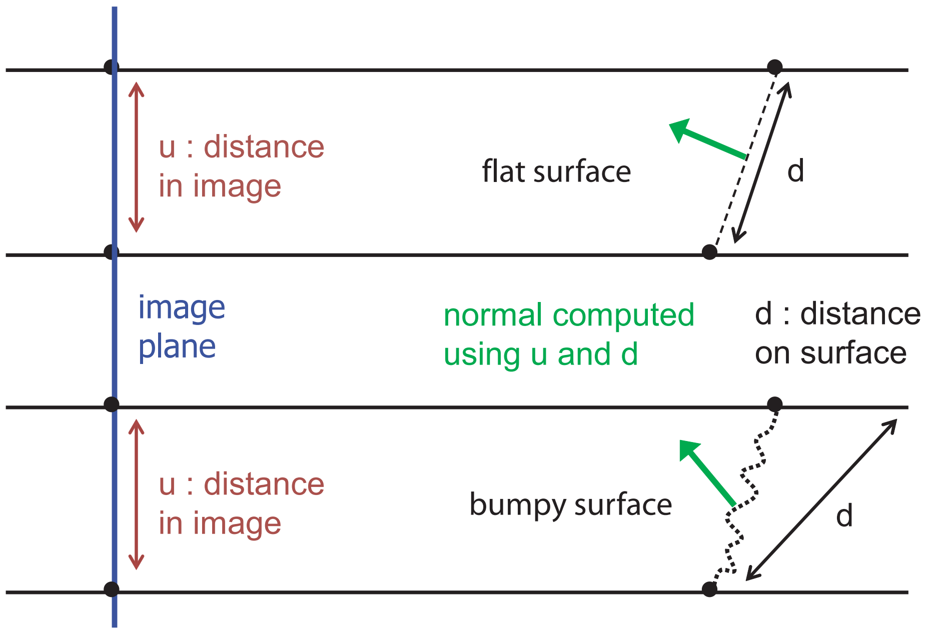


Figure 9.1: When the surface is flat, our monocular reconstruction technique is accurate. However, when the surface is folded or compressed, surface distance estimates are inaccurate estimates of the 3D distance. As a result, normals are exaggerated away from the viewing direction.

9.2.2 Monocular Depth Estimation

our work in monocular depth estimation is most directly limited by the assumptions on the light source: we assume a distant point light source. As mentioned before, because we only derive a single bit from the lighting information, we are highly robust to errors.

The second problem is that geometry that is at a small scale than our texture confounds our normal estimation techniques. Not only is this detail lost, but olds inside the smallest texture elements exaggerate the normals. Specifically, compression of the surface means that the distance in 3D across the texture element is smaller than the distance along the surface. Because normals are computed based on the surface distance, the normals are exaggerated away from the viewing direction. See figure 9.1 for a graphical description of this problem.

9.2.3 Multiview Geometry Estimation

The biggest drawbacks to our multiview geometry estimation is the studio requirements and processing times. Our recording setup requires many cameras – most of the data in this thesis was recording on 8 synchronized video cameras using as many as 7 different high wattage lights. In general, quality improves with the number of cameras and the resolution of the cameras. As a result, we defined **markers per megapixel** as an appropriate way to compare different capture methods.

Second, this approach requires custom garments: we print directly on fabric and then sew together garments. Before we can build 3D models, we need to take photographs of all of the seams. Finally this approach takes 6 to 8 minutes per frame to process. While we believe that these times can be dramatically reduced, it is unlikely that this process will become real-time without significant extra research.

9.2.4 Animating Cloth Using a Database

The biggest drawback to the data driven animation is that it requires a somewhat sizeable database. This can be seen most obviously because animation of walking, running and jogging motions is poor because our capture data does not include walking running or jogging. Second, data with higher resolution folds may not work well: we do not track the motion of individual folds when computing transitions in our database. Future work could

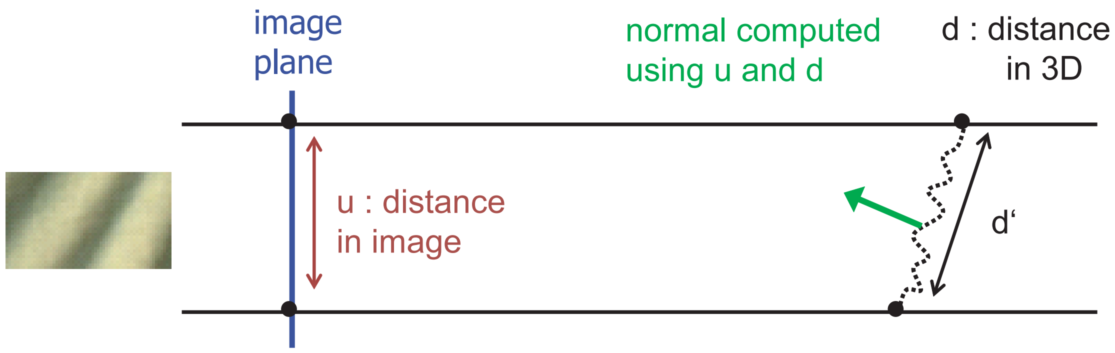


Figure 9.2: We propose a method that uses local shape from shading to account for compression of the surface. This method would require computation of the compression direction first and would result in an estimate of the 3D distance between the two ends of the local patch. This would correct for the exaggerated normals in figure 9.1.

identify a better blending cost.

9.3 Future Work

This section describes a series of projects that have significant potential. We encourage any researchers who have interest in these areas to contact us for further discussion.

9.3.1 Improving the Quality of Monocular Depth Estimation

As mentioned in the previous section, our monocular shape reconstruction doesn't account for small local folds in the surface of the cloth. With additional time, we would pursue a correction factor (figure 9.2) that would take these small folds into account using a local shape from shading model. We would assume a locally extruded surface, compute a compression direction and then estimate surface normals using shape from shading. By integrating these normals, we could compute distance in 3D and make the appropriate corrections to the global normal for the patch.

9.3.2 Extending Monocular Depth Estimation to the Silhouette

Texture and lighting cues are the strongest when the patch is viewed from a nearly frontal direction. However, shape estimation around the contours is just as important. It would be a worthwhile research project to look at ways to combine silhouette extraction techniques with texture and shading techniques. While the quality in these regions would be of a lower quality, the resulting models would be substantially more useful.

9.3.3 Combining Simulation and Capture

Naturally, with a large an established simulation community, it makes sense to combine simulation with capture. One direction for this work is to use a simulator to help with capture. This direction of work has already received some initial interest from [Hasler et al 06]. This approach has two advantages: it intrinsically estimates simulation parameters from video (see next paragraph) and it simplifies the problem: tracking becomes easier because we can predict motion of the cloth. Likewise, results improve because errors are less likely. This is especially true when the correspondence algorithm performs poorly or when significant portions of the cloth are occluded.

However, we believe that the larger application is to train simulators using capture data. Again, there is initial work in this area: [Bhat et al 03]. However, this previous work uses the silhouettes and contours of the cloth to estimate parameters. We believe that a much stronger, more powerful approach comes from estimation of the complete geometry as expressed through a triangulated mesh. Since the representation that we use for a mesh is basically the same as the internal representation used by the simulation community, it is possible to have deep ties between the capture and simulation. One could imagine a calibration setup that uses small pieces of cloth with out custom pattern. A user could print out dozens of such cloth swatches and build up a database of materials. It might even be possible to combine materials to come up with new composite materials. Ideally, this would reduce some of the parameter tweaking necessary to simulate.

9.3.4 Combined Skeleton and Cloth Capture

It would be beneficial to capture both the motion of the cloth and the movement of the underlying body at the same time. This would make it easier and more natural to tie together human motion capture data with cloth capture. However, there are a number of technical hurdles. First, without modification, traditional human motion capture markers are likely to conflict with cloth capture markers. Second, human motion capture markers work best when directly connected to the skeleton, or at least nearby flesh that doesn't move much. It would be difficult to do this and to record the motion of the surface of the cloth: either the cloth motion will be constrained or the skeletal markers will be occluded.

9.3.5 Skin and Skeleton Capture

While all of the examples in this thesis revolve around cloth, any surface that can be printed on can be used for capture. We relied on the fact that the surface not be able to stretch too much, but other surfaces (including skin) maintain this property. It would be relatively easy to print the pattern on spandex and use a spandex suit to capture skin movement.

This approach could also be used in a lower density way to capture human skeleton movement. The larger number of markers would make regression of the skeleton more accurate. However, this approach has several downsides: it requires full camera bandwidth (many mocap systems process and destroy most of the data in the camera) and requires multiple color channels (many mocap systems currently use monochrome image sensors).

9.3.6 Skinning Independently Controlled Characters

One of the major applications of our cloth capture and data driven animation would be to video games. Characters could be skinned as described in chapter 8 after being animated using skeletal motion capture techniques. This would require significant work to speed up the process – the current method takes several minutes to produce a few seconds of animation.

Bibliography

[Anguelov et al 05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. "SCAPE: shape completion and animation of people." In *Proceedings of SIGGRAPH*, 2005.

[Baraff and Witkin 98] David Baraff and Andrew Witkin. "Large Steps in Cloth Simulation." *Computer Graphics*, **32**(Annual Conference Series):43–54, 1998.

[Baraff et al 03] David Baraff, Andrew Witkin, and Michael Kass. "Untangling cloth." *ACM Trans. Graph.*, **22**(3):862–870, 2003.

[Bhat et al 03] K. Bhat, C. D. Twigg, J. K. Hodgins, P. K. Khosla, Z. Popovic, and S. M. Seitz. "Estimating Cloth Simulation Parameters from Video." In *Proc. Symposium on Computer Animation*, 2003.

[Bouguet 05] Jean-Yves Bouguet. "Camera Calibration Toolbox for Matlab.", 2005. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[Bradley et al 05] D. Bradley, G. Roth, and P. Bose. "Augmented Clothing." In *Graphics Interface*, 2005.

[Bridson et al 02] Robert Bridson, Ronald Fedkiw, and John Anderson. "Robust treatment of collisions, contact and friction for cloth animation." In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 594–603, New York, NY, USA, 2002. ACM Press.

[Bridson et al 03] R. Bridson, S. Marino, and R. Fedkiw. "Simulation of clothing with folds and wrinkles." In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pp. 28–36. Eurographics Association, 2003.

[Choe and Kashyap 91] Yoonsik Choe and R. L. Kashyap. "3-D Shape from a Shaded and Textural Surface Image." *IEEE Trans. Pattern Anal. Mach. Intell.*, **13**(9):907–919, 1991.

[Cover and Thomas 91] Thomas M. Cover and Joy A. Thomas. *Information Theory.* John Wiley and Sons, 1991.

[Fang and Hart 04] H. Fang and J. Hart. "Textureshop: Texture Synthesis as a Photograph Editing Tool." In *Proceedings of SIGGRAPH*, 2004.

[Fang and Hart 06] H. Fang and J. Hart. "RotoTexture: Automated Tools for Texturing Raw Video." In *IEEE Transactions on Visualization and Computer Graphics*, 2006.

[Faugeras and Luong 04] Olivier Faugeras and Quang-Tuan Luong. *Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of the Applications*. MIT press, 2004.

[Forsyth and Ponce 02] D.A. Forsyth and J. Ponce. *Computer Vision: a modern approach*. Prentice-Hall, 2002.

[Forsyth and Zisserman 91] D.A. Forsyth and A.P. Zisserman. "Reflections on Shading." *IEEE T. Pattern Analysis and Machine Intelligence*, **13**(7):671–679, 1991.

[Forsyth 01] D.A. Forsyth. "Shape from texture and integrability." In *Int. Conf. on Computer Vision*, pp. 447–452, 2001.

[Forsyth 02] D.A. Forsyth. "Shape from texture without boundaries." In *Proc. ECCV*, volume 3, pp. 225–239, 2002.

[Guskov and Zhukov 02] I. Guskov and L. Zhukov. "Direct Pattern Tracking on Flexible Geometry." In *WSCG*, 2002.

[Guskov et al 03] Igor Guskov, Sergey Klibanov, and Benjamin Bryant. "Trackable Surfaces." In *SCA*, 2003.

[Haddon and Forsyth 97] J. Haddon and D.A. Forsyth. "Shading Primitives." In *Int. Conf. on Computer Vision*, 1997.

[Haddon and Forsyth 98] J. Haddon and D.A. Forsyth. "Shape Descriptions from Shading Primitives." In *European Conference on Computer Vision*, 1998.

[Hartley and Zisserman 00] R. Hartley and A. Zisserman. *Multiple View Geometry.* Cambridge University Press, 2000.

[Hasler et al 06] Nils Hasler, Mark Asbach, Bodo Rosenhahn, Jens-Rainer Ohm, and Hans-Peter Seidel. "Physically Based Tracking of Cloth." In *Proceedings of the 11th International Fall Workshop on Vision, Modeling and Visualization*, 2006.

[Hays et al 06] James H. Hays, Marius Leordeanu, Alexei A. Efros, and Yanxi Liu. "Discovering Texture Regularity as a Higher-Order Correspondence Problem." In *9th European Conference on Computer Vision*, May 2006.

[Heikkila and Silven 97] Janne Heikkila and Olli Silven. "A Four-step Camera Calibration Procedure with Implicit Image Correction." In *Proceedings of Computer Vision and Pattern Recognition*, p. 1106, Washington, DC, USA, 1997. IEEE Computer Society.

[Horn 70] B.K.P. Horn. "Shape from shading : a method for obtaining the shape of a smooth opaque object from one view." Ai tr-232, MIT, 1970.

[House and Breen 00] D.H. House and D. Breen, editors. *Cloth Modelling and Animation.* A.K. Peters, 2000.

[Huang and Lee 89] T.S. Huang and C.H. Lee. "Motion and Structure from Orthographic Projections." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**(5):536 – 540, 1989.

[Kircher and Garland 06] Scott Kircher and Michael Garland. "Editing arbitrarily deforming surface animations." In *Proceedings of SIGGRAPH*, 2006.

[Lin and Liu 06a] W.-C. Lin and Y. Liu. "Tracking dynamic near-regular textures under occlusions and rapid movements." In *Proceedings of the European Conference on Computer Vision*, 2006.

[Lin and Liu 06b] Wen-Chieh Lin and Yanxi Liu. "Tracking Dynamic Near-regular Textures under Occlusion and Rapid Movements." In *Proceedings of the European Conference on Computer Vision*, 2006.

[Lin 05] W-C Lin. *A Lattice-based MRF Model for Dynamic Near-regular Texture Tracking and Manipulation.* PhD thesis, CMU, 2005.

[Lipman et al 05] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. "Linear Rotation-Invariant Coordinates for Meshes." In *SIGGRAPH*, 2005.

[Liu and Lin 03] Y. Liu and W.-C. Lin. "Deformable Texture: the Irregular-Regular-Irregular Cycle." In *The 3rd intl. workshop on texture analysis and synthesis*, pp. 65–70, 2003.

[Liu et al 04] Y. Liu, W.-C. Lin, and J. Hays. "Near-Regular Texture Analysis and Manipulation." *ACM Transactions on Graphics*, **23**(3):366–374, 2004.

[Lobay and Forsyth 04] Anthony Lobay and D.A. Forsyth. "Recovering shape and irradiance maps from rich dense texton fields." In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2004.

[Lobay and Forsyth 06] Anthony Lobay and D. A. Forsyth. "Shape from Texture without Boundaries." *International Journal of Computer Vision*, **67**(1):71–91, 2006.

[Loh and Hartley 05] Angeline M. Loh and Richard Hartley. "Shape from non-homogeneous, non-stationary, anisotropic, perspective texture." In *Proceedings of the British Machine Vision Conference*, 2005.

[Lowe 04] D.G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints." *IJCV*, 2004.

[Park and Hodgins 06] Sang Il Park and Jessica K. Hodgins. “Capturing and animating skin deformation in human motion.” In *Proceedings of SIGGRAPH*, 2006.

[Perona and Malik 90] P. Perona and J. Malik. “Scale-Space and Edge Detection Using Anisotropic Diffusion.” *PAMI*, 1990.

[Pilet et al 05] Julien Pilet, Vincent Lepetit, and Pascal Fua. “Real-time Non-Rigid Surface Detection.” In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2005.

[Pollefeys et al 99] Marc Pollefeys, Reinhard Koch, and Luc Van Gool. “Self-Calibration and Metric Reconstruction Inspite of Varying and Unknown Intrinsic Camera Parameters.” *Int. J. Comput. Vision*, **32**(1):7–25, 1999.

[Pritchard and Heidrich 03] D. Pritchard and W. Heidrich. “Cloth motion capture.” *Eurographics*, 2003.

[Provot 95] Xavier Provot. “Deformation constraints in a mass-spring model to describe rigid cloth behavior.” In *Graphics Interface*, 1995.

[Savarese et al 04] S. Savarese, M. Chen, and P. Perona. “Recovering local shape of a mirror surface from reflection of a regular grid.” In *Proceedings of the European Conference on Computer Vision*, 2004.

[Scholz and Magnor 04] V. Scholz and Marcus A. Magnor. "Cloth Motion from Optical Flow." In *VMV*, 2004.

[Scholz and Magnor 06] V. Scholz and M. Magnor. "Texture Replacement of Garments in Monocular Video Sequences." In *Rendering Techniques*, 2006.

[Scholz et al 05] V. Scholz, T. Stich, M. Keckeisen, M. Wacker, and M. Magnor. "Garment Motion Capture Using Color-Coded Patterns." In *Eurographics*, 2005.

[Sorkine et al 04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. "Laplacian Surface Editing." In *Symposium of Geometry Processing*, 2004.

[Sumner and Popovic 04] R.W. Sumner and J. Popović. "Deformation Transfer for Triangle Meshes." *ACM Transactions on Graphics*, **23**(3):397–403, 2004.

[Sumner et al 05] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popovic. "Mesh-based inverse kinematics." In *Proceedings of SIGGRAPH*, 2005.

[Tanie et al 05] H. Tanie, K. Yamane, and Y. Nakamura. "High Marker Density Motion Capture by Retroreflective Mesh Suit." In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[Terzopolous et al 87] D. Terzopolous, J. Platt, A. Barr, and K. Fleischer. "Elastically deformable models." *Computer Graphics (SIGGRAPH 87 Proceedings)*, pp. 205–214, 1987.

[Tomasi and Kanade 92] Carlo Tomasi and Takeo Kanade. "Shape and motion from image streams under orthography: a factorization method." *Int. J. Comput. Vision*, **9**(2):137–154, 1992.

[Triggs et al 00] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. "Bundle Adjustment - A Modern Synthesis." In *ICCV '99: Proceedings of the International Workshop on Vision Algorithms*, pp. 298–372. Springer-Verlag, 2000.

[Tsai 92] Roger Y. Tsai. "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses." *Radiometry*, 1992.

[Tsap et al 00] Leonid V. Tsap, Dmitry B. Goldgof, and Sudeep Sarkar. "Nonrigid Motion Analysis Based on Dynamic Refinement of Finite Element Models." *IEEE Trans. Pattern Anal. Mach. Intell.*, **22**(5):526–543, 2000.

[Ullman 79] S. Ullman. *The Interpretation of Visual Motion.* MIT press, 1979.

[White and Forsyth 05] Ryan White and David A. Forsyth. "Deforming Objects

Provide Better Camera Calibration." Technical Report UCB/EECS-2005-3, U.C., Berkeley, 2005.

[White and Forsyth 06a] Ryan White and David A. Forsyth. "Combining Cues: Shape from Shading and Texture." In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[White and Forsyth 06b] Ryan White and David A. Forsyth. "Retexturing Single Views Using Texture and Shading." In *Proceedings of the European Conference on Computer Vision*, 2006.

[White et al 05] Ryan White, Anthony Lobay, and David A. Forsyth. "Cloth Capture." Technical Report UCB/CSD-5-1387, U.C., Berkeley, 2005.

[White et al 06] Ryan White, David A. Forsyth, and Jai Vasanth. "Capturing Real Folds in Cloth." Technical Report UCB/EECS-2006-10, U.C., Berkeley, 2006.

[White et al 07] Ryan White, Keenan Crane, and David Forsyth. "Capturing and Animating Occluded Cloth." In *Proceedings of SIGGRAPH*, 2007.

[Zelinka et al 05] S. Zelinka, H. Fang, M. Garland, and J. Hart. "Interactive Material Replacement in Photographs." In *Proceedings of Graphics Interface*, 2005.

[Zhang 00] Zhengyou Zhang. “A Flexible New Technique for Camera Calibration.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(11):1330–1334, 2000.

[Zhang et al 99] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. “Shape from Shading: A Survey.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21**(8):690–706, 1999.